UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Mejora del aplicativo web de una entidad bancaria usando diversas tecnologías relacionadas con Java

Autor:
Yun Wu Xu

Directora:
Monica Raquel Herrera Jimenez

Ponente:
Jaime.M Delgado

Especialidad:
Tecnologías de la Información

Junio 2020

Trabajo Fin de Grado

Resumen

Una de las principales funciones de una empresa bancaria es poder ofrecer a sus clientes un ventanal de funciones tal que se sientan cómodos y no tengan problemas a la hora de gestionar cualquier problema a través de la aplicación web. En estos tiempos, con la rápida evolución que de las tecnologías, es muy importante estar a a la vanguardia de estas.

Con el desarrollo conseguimos poder mejorar tanto la parte funcional de la aplicación como la parte de la implementación. Además nos podremos centrar en lo que significa la experiencia del usuario, algo importante en lo que respecta a la parte web de un desarrollo.

Por otra parte, aprenderemos a trabajar en equipo, ya que cada célula tiene asignada una parte del proyecto en general y como al fin y al cabo dependeremos de otras operaciones tendremos que tratar este tema.

Por último, en este proyecto se consigue aprender a como mejorar la implementación que hay detrás para poder ofrecer un buen servicio debido que programar todo pueden hacerlo, pero programar bien no.

Resum

Una de les principals funcions d'una empresa bancària és poder oferir als seus clients un finestral de funcions tal que se sentin còmodes i no tinguin problemes a l'hora de gestionar qualsevol problema a través de l'aplicació web. En aquests temps, amb la ràpida evolució que de les tecnologies, és molt important estar a a l'avantguarda d'aquestes.

Amb el desenvolupament aconseguim poder millorar tant la part funcional de l'aplicació com la part de la implementació. A més ens podrem centrar en el que significa l'experiència d'usuari, una cosa important pel que fa a la part web d'un desenvolupament.

D'altra banda, aprendrem a treballar en equip, ja que cada cèl·lula té assignada una part de el projecte en general i com a tal i a el cap dependrem d'altres operacions haurem de tractar aquest tema.

Finalment, en aquest projecte s'aconsegueix aprendre a com millorar la implementació que hi ha darrere per poder oferir un bon servei a causa de programar tot poden fer-ho, però programar bé no.

Abstract

One of the main functions of a banking company is to be able to offer its customers a amount of functions so that they feel comfortable and have no problems managing any problem through the web application. Nowadays, with the rapid evolution of technologies, it is very important to be at the forefront of these.

With development we are able to improve both the functional part of the application and the implementation part. In addition we can focus on what the user experience means, something important in regards to the web part of a development.

On the other hand, we will learn to work as a team, since each cell has been assigned a part of the project in general and since, after all, we will depend on other operations, we will have to deal with this issue.

Finally, in this project it is possible to learn how to improve the implementation behind it in order to offer a good service because programming can do everything, but not programming well.

Índice

1. Contexto y Alcance

1.1. Introducción

Toda grande compañía necesita que su producto, el que quiere ofrecer a sus clientes, sea positivo, consistente, predecible y deseable. Es decir, buscar que al usuario le sea muy útil y cómodo el uso del servicio en cuestión y que tenga ganas de volver a utilizarlo o recomendarlo.

Hoy en día, con el avance de las tecnologías, cada vez hay más herramientas que cualquier persona puede utilizar para que su servicio sea lo "más bonito" de todo, sí que juega un papel valioso, pero no siempre es más importante lo del exterior, sino también cuenta mucho el interior.

A parte, se añade una forma de trabajar dividido en sprints [1], no me refiero en la parte novedosa en la sociedad, sino dentro de este departamento. Con esto conseguimos que durante un cierto periodo de tiempo, en este caso 2 semanas, podamos realizar ciertas tareas y descubrir si hay problemas durante el desarrollo, dudas, mejoras,...

Por consiguiente, con este trabajo se pretende que el aplicativo web de una gran empresa bancaria mejore tanto en las funcionalidades de sus prestaciones como también en la parte estética de ellas. Además, con la metodología que se utiliza, podemos estar en constante contacto con el cliente y así poder tener una visión clara de cómo está el producto en cuestión.

1.2. Términos & Conceptos

Ahora pasaré a comentar algunas terminologías que sería bueno tener en mente su significado para el resto del documento, ya que me referiré a ellas más adelante:

- **US (User Stories)** [2]: Hace referencia a las pequeñas tareas en las que se divide el producto en general, para que cada uno del equipo se encargue de una de ellas independiente del resto, ya que luego se pondrá todo en común.
- **Nice to have**: Son, al fin y al cabo, una serie de US, pero que tienen la finalidad, de que si durante un sprint se logra el objetivo de acabar todas las US que se habían pedido, entonces se puede intentar hacer estas o tenerlas bastante avanzadas para el siguiente sprint.
- **Weblogic** [3]: Es un servidor de aplicaciones que se utiliza para probar el desarrollo de nuestro aplicativo y probar todo tipo de servicio.

- **Scrum** [4]: Es un método para trabajar en equipo a partir de iteraciones o sprints. Así pues, scrum es una metodología ágil, por lo que su objetivo será controlar y planificar proyectos con un gran volumen de cambios de última hora, en donde la incertidumbre sea elevada.
- **Sprint** : Sprint es el nombre que va a recibir cada uno de los ciclos o iteraciones que vamos a tener dentro de un proyecto scrum. La idea que hay detrás de este, es que si hay un proyecto bastante largo, vamos a poder dividir a este mismo en doce sprints que tienen una duración de dos semanas cada uno, eso conlleva dos sprints por mes. En cada uno de estos vamos a ir consiguiendo un producto, que siempre, y esto es muy importante, sea un producto que esté funcionando.
- **Guindo**: Manual de estilos que es proporcionado por la entidad bancaria en la que estamos haciendo el proyecto.

1.3. Problema a resolver

Actualmente existen bastantes entidades bancarias que tienen una gran reputación, a nivel estatal, donde mucha población (España) son clientes de estas. Pero claramente hay mucha presión entre bancos para intentar "recolectar" el mayor número de usuarios posibles, ya sean de otras entidades o nuevos usuarios que quieren abrir una nueva cuenta corriente.

Es por ello que muchas de estas empresas están cada día, en busca de alguna mejora, solución,... para que la mayor parte de sus clientes, ya sean antiguos o aquellos que quieren unirse como nuevo miembro, utilicen sus productos, de esta manera podremos tener a los usuarios contentos y así aumentarán su confianza, ya que a medida que vaya en alza, menos motivos tendrá de querer cambiar de empresa [5].

A su vez, tampoco es cuestión de quedarse atrás en lo tecnológico, hay que mantenerse en la vanguardia de la tecnología y es por eso que se tiene que mejorar tanto en la parte exterior, como en la parte interior de la aplicación de esta. Por muy bonito que sea es posible que sea muy poco intuitivo, que tenga muy mala seguridad, poca eficacia, ... Es por eso que nunca está de más hacerle un buen lavado de cara.

Con la intención de mejorar cada uno de los puntos tecnológicos de la empresa [6], con este proyecto lo que se consigue es esto. La ayuda que da la tecnología en este tipo de ámbito, y también en cualquiera, suponen una gran ventaja comparado con las desventajas que puedan aparecer.

Además las empresas desean una actualización de los sistemas ya que, hay algunas multinacionales que no se han esforzado en la parte tecnológica de sus compañías. Por eso es un buen punto a favor que sus rivales hayan apostado

por esta parte (cabe recordar que no me refiero únicamente en lo estético de las aplicaciones, sino en la lógicas e implementaciones de estas).

1.4. Actores implicados

En este apartado explicaré qué tipos de actores aparecen en este trabajo y cuales son las funciones de cada uno de ellos:

- **Desarrollador web** [7]: Es el profesional que se encarga de crear, a partir de un lenguaje de programación (Java y JavaScript) las páginas webs de la empresa. Otra de sus funcionalidades es la de programar todas las acciones que conllevan esta web y la de implementar la parte de interfaz del usuario.
- **Analista programador** [8] : Es el encargado de realizar las funciones de un analista técnico y de un programador; es decir, parte de una información previa recibida del analista funcional, en función de la cual desarrolla las aplicaciones y organiza los datos.
- **Usuario Final** : Cada una de las personas a las cuales se dirige este producto, de una manera más indirecta, ya que finalmente seremos todos nosotros, como usuarios del banco, quienes utilizaremos la aplicación día tras día, y seremos nosotros quienes juzguemos todo su funcionamiento. Ellos también serán los beneficiarios de este trabajo.
- **Cliente Directo**: Es la empresa a la cual ofrecemos nuestros servicios para mejorar su aplicativo web. Son ellos a los que va dirigido nuestro desarrollo y a la vez también a los que les beneficiará.

2. Justificación

Hoy en día existe una cosa muy clara en el ámbito de la informática, por muy complicada que se pueda dar una situación en concreto, hay una probabilidad muy alta de que alguien se haya enfrentado al mismo problema o uno muy parecido. No quiero decir que sean idénticas, pero sí del mismo estilo y se pueda moldear a tu problema.

Por consiguiente, con esto en mente, se dice que existe un posible patrón [9] en el momento en el que la forma de resolver el problema se puede extraer, explicar y reutilizar en diferentes ámbitos.

2.1. Soluciones Posibles

Existen diferentes patrones en los cuales podemos diseñar/implementar nuestro proyecto, pero los que comentaré a continuación son los que más se utilizan:

2.1.1. Patrón de capas

Es un patrón [10] bastante común el cual, se divide en distintas capas tal que cada capa engloba las tareas que contengan características en común.

Por lo general, se suelen usar 3 capas, pero se pueden subdividir en más que corresponden a:

- capa de presentación: en esta capa predomina todo aquello que tenga interacción con el usuario.
- capa de dominio: está formada por las entidades, que representan objetos que van a ser manejados o utilizados por toda la aplicación.
- capa de persistencia: contienen aquellas clases que tienen acceso a la base de datos, utilizando los procedimientos necesarios para el almacenamiento y la búsqueda de la información requerida.

Por último, decir que cada una de estas capas proporcionan servicios a la capa superior, por ejemplo la capa de persistencia da soporte a la capa de dominio que está a su vez le da a la capa de presentación.

2.1.2. Patrón cliente-servidor

Este patrón [11] tiene dos partes definidas, una es la parte de cliente, que es el que envía todas las peticiones que necesita, y la otra la compone un servidor, que este esta de manera pasiva en un puerto ya asignado donde recibirá las peticiones que les envía los clientes conectados en un puerto arbitrario para su comunicación.

Los clientes y los servidores pueden estar conectados a una red local o una red amplia, como la que se puede implementar en una empresa o a una red mundial como lo es Internet.

Bajo este modelo cada usuario tiene la libertad de obtener la información que requiera en un momento dado proveniente de una o varias fuentes locales o distantes y de procesarla según le convenga. Los distintos servidores también pueden intercambiar información dentro de esta arquitectura.

2.1.3. Modelo-vista-controlador

Es un patrón [12] en el cual separa los datos de la aplicación, con la interfaz de usuario y la lógica de control en los tres componentes distintos. Es bastante similar al patrón de capas, de 3 capas, pero tiene una curiosidad. Este puede programar en 3 capas pero sin seguir dichas reglas, pero en cambio, al revés, no es la misma división que propone MVC.

En lo que respecta a los componentes, tiene 3 partes:

- el modelo: es el objeto que representa los datos del programa, contiene las funcionalidades y los datos básicos
- la vista: es el objeto que maneja la presentación visual del modelo, es decir muestra la información al usuario
- el controlador: es el objeto que proporciona sentido a las peticiones del usuario, actuando sobre los datos representados por el modelo. También se encarga de realizar los cambios pertinentes ya sea cuando haya cambio de vistas o cuando haya cambios de datos.

Esto se utiliza para separar las representaciones internas de información, de las formas en que se presenta y acepta la información del usuario. Dicho de otra manera, desacopla los componentes y permite la reutilización eficiente del código.

2.2. Solución propuesta

Después de analizar las principales propuestas que hay hoy en día en la actualidad, decidimos escoger una que fuera aventajada a la hora de usar aplicaciones World Wide Web, ya que es nuestro caso.

Decidimos escoger la arquitectura MVC, debido a que uno de sus principales usos es para este tipo de aplicaciones, entre los principales lenguajes de programación. Aparte, otro de los motivos, es que a la hora de agregar vistas (como es en este caso), las engloba todas en la capa de la Vista, como resultado podremos separar la lógica de negocio con la de presentación.

Tener una capa de controlador, nos facilita los cambios de vistas y las peticiones a la capa de modelo. Tener un controlador es bastante útil a la hora de manejar la gran cantidad de ventanas que tenemos y mostramos. También cuando hablamos sobre la parte de los datos, con la ayuda del controlador sabremos a qué servicio debemos pedir la petición e ir directamente.

Por último, nos favorece a la hora de reutilizar los componentes que hayamos utilizado anteriormente.

3. Alcance

3.1. Objetivos

El objetivo principal de este proyecto, es la mejora de la experiencia del usuario en el uso de la aplicación de una entidad bancaria para que un usuario cualquiera, ya pueda ser un señor mayor o como un adolescente, puedan utilizar de la mejor manera posible la aplicación. Por otra lado, en lo que respecta a la usabilidad, no debería ser muy compleja, ya que no tendría mucho sentido tener que esconder algunas funciones que les ayuden.

Para implementar este trabajo, la principal herramienta que se utilizará será el lenguaje de programación Java y Javascript. Por otra parte, se utilizará un manual de estilos, llamado guindo que nos proporciona todos los estilos que son necesarios para cada uno de los componentes que aparecen en las páginas. De esta manera, si hay que modificar lo estilos de todas las cabeceras, por ejemplo, únicamente tendremos que modificar ese fichero. Así lo tendremos de una manera centralizada.

El proyecto finalizará con la mejora de cada una de estas operativas funcionales propuesta por la entidad bancaria y estarán disponibles en la aplicación web de esta misma a disposición de cualquier usuario.

3.2. Requisitos funcionales

Las distintas funcionalidades que se deben tener en cuenta son las siguientes:

- Permitir al usuario acceder a todas las funcionalidades de manera eficaz, sin tener que realizar un largo tiempo de espera. Es por ello que intentaremos que las llamadas a back-end se hagan todas al mismo tiempo.
- No mostrar de ninguna manera la información relevante del usuario , ya sea a través de la mensajería que corre por detrás.
- La aplicación permitirá tener el control de las operativas más usadas por parte de los usuarios. Con ello tendrá una lista de las funcionalidades más utilizadas o mas visitadas por parte del público. Algo importante en este tipo de ámbito.
- Mejora visual de sus componentes y su actualización. Sobre este tema hay que poner más hincapié, sin olvidar las demás.
- Lógica detrás de las operaciones, debido a que sino cualquier usuario podría realizar acciones no pertinentes.

3.3. Requisitos no funcionales

A parte de todos aquellos requisitos que afectan directamente a la aplicación, ahora comentaré aquellos que no afectan, pero que se deben tener en cuenta a la hora de hacer el desarrollo:

- **Fiable**: Los datos que rellene el usuario o los que desea consultar, en ningún caso se deben perder. Si no tuviéramos este requisito, no tendríamos la confianza del usuario y esto es lo más importante para nosotros ya que son a ellos a los que va dirigido nuestro proyecto.
- **Seguro**: Una de las características más importantes para realizar este proyecto dirigido a las entidades bancarias, es que sea fiable. Circula mucha información valiosa y los usuarios no querrán que esta pueda acabar en manos de terceras partes. Por ello este requisito debería ser el más importante a cumplir.
  Por otra parte, únicamente los usuarios con cierto nivel de privilegios (esto ya dado por el organismo empresarial), podrá realizar ciertos tipos de operaciones.
- **Usable**: El usuario deberá tener la mayor facilidad en el uso del aplicativo, es decir, no deberá tener dificultades a la hora de realizar una operación y en caso contrario deberá tener una ayuda por si hay algo que no entendiera o un mensaje de error explicando lo sucedido.
- **Accesible**: Facilitar que el usuario pueda acceder a la aplicación por medio de cualquier dispositivo o navegador, nuestra intención no debe ser nunca obligar a ningún miembro a usar un cierto portal que no le sea de su agrado.

3.4. Obstáculos y riesgos

Para finalizar, también hay que tener en cuenta una serie de riesgos que pueden surgir durante el transcurso del proyecto:

- **Limitación del tiempo**: Ya sea por el tiempo que hay entre entregas y los problemas que te puedas encontrar de las anteriores versiones, el riesgo en tomar decisiones que no sean las más eficientes posibles, es alto. Cuando no hay tiempo, cualquier idea que resuelva el problema es bien recibida y no debería ser así. Es por ello que la planificación deberá ser precisa, teniendo tiempo para todo y teniendo que cumplirla para que todo termine como se desea.
- **Inexperiencia en el tema de usabilidad**: No tener noción en la experiencia de la usabilidad de los productos es algo que nos perjudica directamente, ya que creemos que es intuitivo para nosotros pero para el resto de los usuarios a lo mejor no lo es.

- **Problemas con los servicios**: Al tener que utilizar servicios de otros proyectos y no tener la documentación, es muy difícil poder entender la lógica que han utilizado los otros desarrolladores. Esto provocaría una gran pérdida de tiempo, ya que se debería contactar con el equipo que hubiera hecho ese servicio y de esta manera intentar entender lo básico para poder brindarnos la ayuda necesaria.

4. Metodología y Rigor

Un tema fundamental, es escoger qué metodología seguirá nuestro trabajo, ya que hay distintas maneras de planearlo. Además, comentaré las herramientas que se utilizarán para inspeccionar la calidad del código, detectar los errores y las vulnerabilidades.

4.1. Metodología

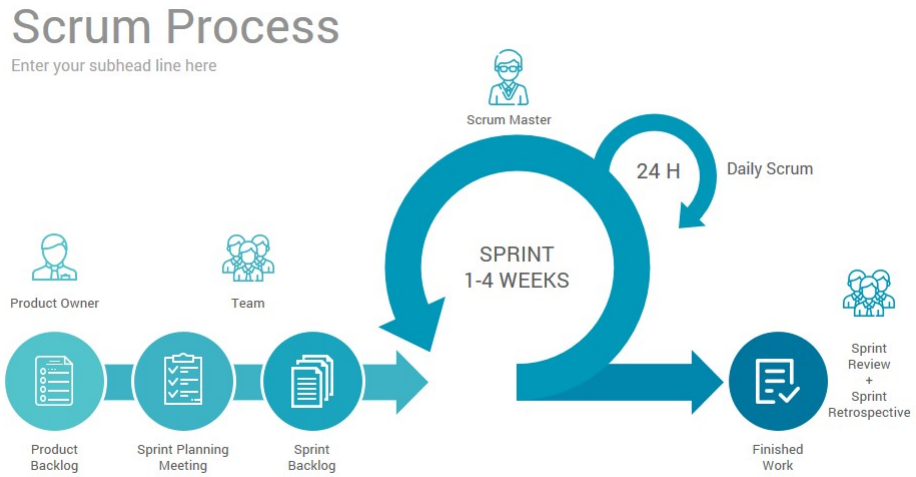


Figura 1: Ejemplo metodología Scrum

La metodología que se aplicará para este proyecto será scrum, como se puede apreciar en la Figura 1. Su finalidad como he mencionado anteriormente, será la de controlar y planificar proyectos grandes.

Lo más habitual es planificarlo por semanas y en nuestro caso es así, cada 2 semanas. Cada una de estas planificaciones son llamadas sprints o iteraciones. En cada una de estas iteraciones se realiza un cierto número de tareas que son asignadas a partir de un sprint planning que se realiza al inicio, o al final, depende de como se mire, donde se priorizará las actividades que sean de interés inmediato. Por otra parte, también se revisará el trabajo validado del anterior Sprint.

Aparte, se realizarán “Dailies o scrum diario ” donde se irá comentando lo que se ha realizado durante el día anterior y si hay algún inconveniente,

expresarlo. Es una manera de tener el control del proyecto y saber si se podrán cumplir las tareas de este Sprint. Tiene mucha utilidad en los proyectos de este calibre, ya que permite ir controlando la realización de los encargos y poder ver si hay contingencias o imprevistos, como comentaré más adelante.

Por último, para mejorar el ámbito de trabajo y la cohesión del grupo existen las retrospectivas donde el equipo de desarrollo puede proponer cambios o ideas para mejorar el proceso de trabajo y así para los siguientes ciclos poder aplicarlos y mejorarlos. Es una manera de fomentar la cohesión del grupo, siendo un papel fundamental en cualquier trabajo.

4.2. Herramientas de Validación

Por lo que respecta a la hora de validar el código o encontrar errores, vulnerabilidades, ... tenemos dos herramientas esenciales para su uso.

4.2.1. Jenkins

Es un servidor [13] de integración continua, en pocas palabras, consiste en compilar y ejecutar todo un proyecto para que no aparezca ningún error posteriormente, gratuito y open-source. Ciertamente, se recomienda que este proceso se ejecute cada cierto tiempo, aprovechando que es una metodología scrum, se realizará cada vez que se acabe una tarea. Una de las ventajas de utilizar un software de integración continua es que los desarrolladores pueden detectar y solucionar problemas de integración de forma continua , evitando el caos de última hora cuando se acercan las fechas de entrega del proyecto en global.

Lo que ejecuta jenkins es el proyecto,código,..., y lo pasa por diferentes fases, desde el build, hasta test unitarios, deployment.

4.2.2. Sonar

Es una herramienta que permite realizar análisis estático de código fuente de manera automática. A su vez, también busca patrones con error, malas prácticas o incidentes.

Dicho de otra manera, comprueba que el código a revisar sea robusto y eficiente, ya que hoy en día es muy fácil hacer que funcione, pero hay que conseguir que el código sea la mejor versión posible.

Hay tres niveles distintos de issues (problemas a resolver), el minor que es algo de poca importancia, el mayor, el cual no debería haber ninguno, y el extrem, que junto al mayor son los dos tipos de errores que ensucian nuestro código. Esto tiene importancia, ya que el Jenkins también lo pasa por el Sonar y comprueba que la suma de los issues mayor y extrem sean menores de 10 para considerar el proyecto como válido.

5. Descripción de las tareas

A continuación explicaré las tareas a realizar, donde en cada una de estas explicaré que se debe hacer, quien la realizará, los recursos tanto humanos como materiales y los que son relacionados con los gastos generales (electricidad, internet,...).

Aclarar que cuando hablo sobre el ordenador, en referencia a los recursos hardware, se incluyen todos los recursos hardware y software (que los explicaré más adelante). Además, en referencia a la oficina, este ya incluye todas los gastos generales, comentadas más adelante también.

5.1. Tareas

Explicaré cada una de las tareas a realizar en este proyecto. Este se ha dividido de manera que haya sprints cada 2 semanas, que esto supone un total de 2 al mes. Desarrollaré en los siguientes apartados el conjunto global de las tareas, pero a la hora de generar el gantt los dividiré por los sprints correspondientes.

La secuencia lógica a seguir es la siguiente. Siempre, en cada sprint habrá una parte que sea tarea de análisis y acto seguido desarrollo. Estas serán realizadas siguiendo este orden, por lo cual, cualquier tarea dentro de la fase de análisis será precedente de la fase desarrollo. Por tanto, una dependencia clara de la etapa de desarrollo es la de análisis. Por último, las tareas de landing (nuevas vistas) siempre serán las precedentes de todos los otros componentes, de las páginas, a implementar.

Por lo que respecta a la duración total, empieza desde el día 17 de febrero del 2020 hasta la entrega de la memoria, que corresponde al 2 de junio de 2020. Quitando el primer mes, dedicado a la primera tarea, entonces nos restan 3 meses con una media de 40 horas de trabajo a la semana. El total de horas en global es de 661.

5.1.1. Gestión de proyecto

- Daremos a conocer el contexto y el alcance del proyecto, donde se hará una búsqueda de la información y las posibles soluciones que hay en la actualidad. Además, se hablará de la metodología a utilizar y la justificación de esta decisión. Por otra parte, también se explicará un conjunto de herramientas a utilizar a la hora de realizar el trabajo.

- Una planificación temporal, donde se exponen todas las tareas a realizar y sus respectivas explicaciones, el diagrama de gantt donde se mostrará el tiempo de dedicación (de manera gráfica) prevista para cada una de las fases y los riesgos que puede suponer.
- Un presupuesto del proyecto en global y un pequeño informe de sostenibilidad.
- Un documento final que engloba los tres primeros puntos con sus respectivas correcciones y mejoras.
- El total de horas estimadas para esta tarea es de: 63 horas, aproximadamente.
- Los recursos necesarios para esta fase son:
  - Humanos: Tutor de proyecto
  - Materiales: Ordenador, oficina.

5.1.2. Análisis

- Estudio de los casos repetidos. Nos centraremos en hacer una búsqueda dentro del proyecto global, si hay otra célula que haya realizado una operación similar, se tendría que estudiar el caso para su posible reutilización del código.
- Estudiar si es una nueva prestación, un nuevo servicio al cliente.
- Definición de las funcionalidades. Se asignan qué tipo de funcionalidades serán necesarias con las necesidades que tiene el cliente.
- Definición de las tecnologías a utilizar. Selección de las tecnologías necesarias según las funcionalidades obtenidas anteriormente.
- Lógica de negocio (reglas) a seguir.
- Casos de usos de cada operativa a realizar.
- Los recursos necesarios para esta fase son:
  - Humanos: Analista/Analista Programador .
  - Materiales: Ordenador, repositorio (el almacenamiento en la nube para tener control de las subidas y las versiones),oficina.

5.1.3. Desarrollo de las funcionalidades

- Desarrollo de las funcionalidades encontradas y tratadas en la parte de análisis.
- Se implementan todo tipo de componentes: menús, enlaces, botones, pestañas, landings,....
- Incluye todo tipo de cambios, renovación,... a nuevos servicios, o ya preestablecidos, de las operativas (en caso que sea necesario).
- Se prueba que todo los cambios realizados funcionen correctamente, y que sigan la lógica anterior o una nueva.
- Los recursos necesarios para esta fase son:
  - Humanos: Equipo de Desarrollo.
  - Materiales: Ordenador,servidores,repositorio,oficina.

5.1.4. Documentación

- Redacción de la memoria del proyecto final.
- Elaboración de una presentación donde se expondrá a un público y un tribunal.
- Los recursos necesarios para esta fase son:
  - Humanos: Tutor del proyecto.
  - Materiales: Ordenador, oficina.

5.1.5. Reuniones Scrum

- Sprint Planning: Donde se pactan las tareas a realizar durante el sprint en cuestión.
- Restrospectiva: Donde se realiza un trabajo en equipo para averiguar en que se puede mejorar, ya sea el equipo, la relación con el cliente,...
- Review: Donde se hace una revisión sobre las tareas del anterior sprint.
- Los recursos necesarios para esta fase son:
  - Humanos: Equipo Scrum.
  - Materiales: Ordenador. oficina.

5.2. Resumen

Código	Nombre	Tiempo	Dependencia
T0	Gestión de proyecto	63 h	
T0.1	Contexto y alcance	25 h	
T0.2	Planificación temporal	9 h	T0.1
T0.3	Presupuestos	10 h	T0.2
T0.4	Documento final	19 h	T0.3
T1	Análisis	150 h	T0.4
T1.1	Casos repetidos	5 h	
T1.2	Nuevas Prestaciones	5 h	
T1.3	Funcionalidades + Tecnologías	50 h	T1.2 (opcional)
T1.4	Lógica	60 h	T1.3
T1.5	Casos de Uso	30 h	T1.4
T2	Desarrollo e implementación	330 h	T1
T2.1	Desarrollo de las funcionalidades	130 h	T1.3
T2.2	Implementación de componentes	130 h	T2.1
T2.3	Nuevos servicios	30 h	T2.2
T2.4	Testing	40 h	T2.3
T3	Documentación	100 h	
T3.1	Memoria	70 h	
T3.2	Defensa TFG	30 h	T3.1
T4	Reuniones Scrum	18 h	
T4.1	Restrospectiva, Planning, Review	18 h	

Tabla 1: Resumen Tareas

6. Estimación y Gantt

6.1. Estimación

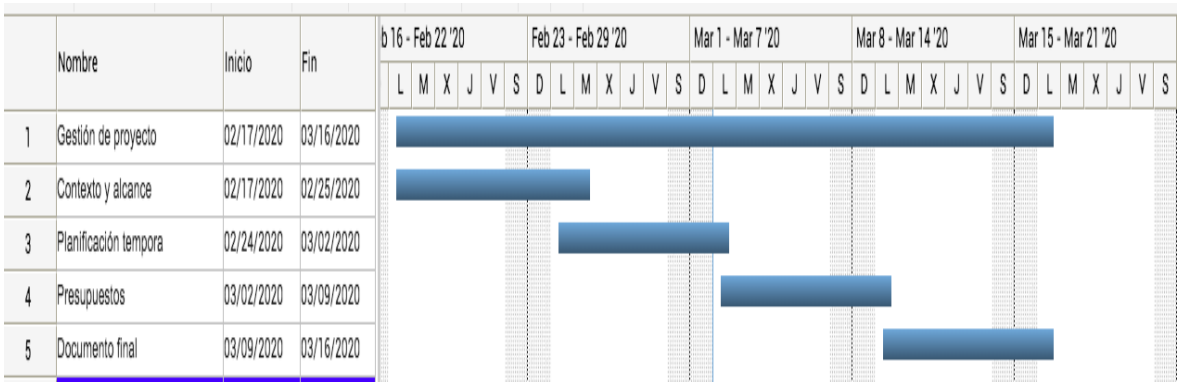
Todas las estimaciones se pueden ver en la tabla de arriba, en el apartado 5.2 Resumen. Las horas son tan altas por el motivo que he expuesto anteriormente. Si fuera menor, claramente las horas impuestas en la tabla bajarían linealmente con respecto a las horas que se trabajaría a la semana, pero cumpliendo el mínimo necesario que es de 540 horas aproximadamente.

Se le han asignado las altas horas en el desarrollo y la implementación debido que es la tarea principal de este proyecto. Todo el peso está en esta etapa del proceso, ya que al tener que volver a reconstruir la página y/o los componentes de esta. Además, al no tener experiencia de aplicaciones web, será necesario gastar más horas en este aspecto. Por otra lado, a la hora de asignar las tareas, agrupo todos los componentes (sea menús, enlaces,...) en la misma vista y podemos apreciar que la idea es la misma pero a la hora de implementarla es distinta.

Las horas propuestas en los análisis, he creído invertir un cierto número de horas elevadas, a la vista esta que es una etapa bastante importante, que nos podrá ahorrar tiempo en el futuro y nos permitirá tener más claro por dónde avanzar. En en apartado de "nuevas propuestas" se invierte poco tiempo debido a que solamente es investigar las operativas que hay y si se pueden reutilizar de otras para la nueva. En caso negativo, se pasará a la fase de "funcionalidades", por eso he añadido una "dependencia opcional".

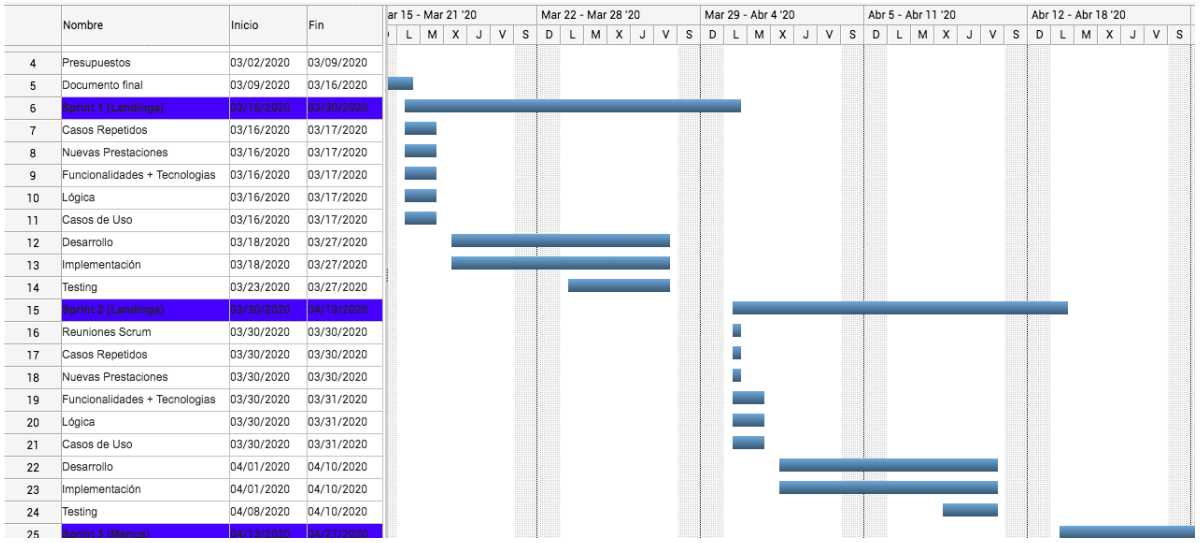
Al ser una primera aproximación, las horas repartidas podrán ir variando respecto al de ahora, lo más seguro (si se diera el caso) es que se podrían acabar añadiendo en el apartado de análisis. A parte, las horas de testing podrían aumentar, ya que son vitales para poder mostrar las evidencias y, aún más, para mostrarlo al consumidor final.

6.2. Gantt



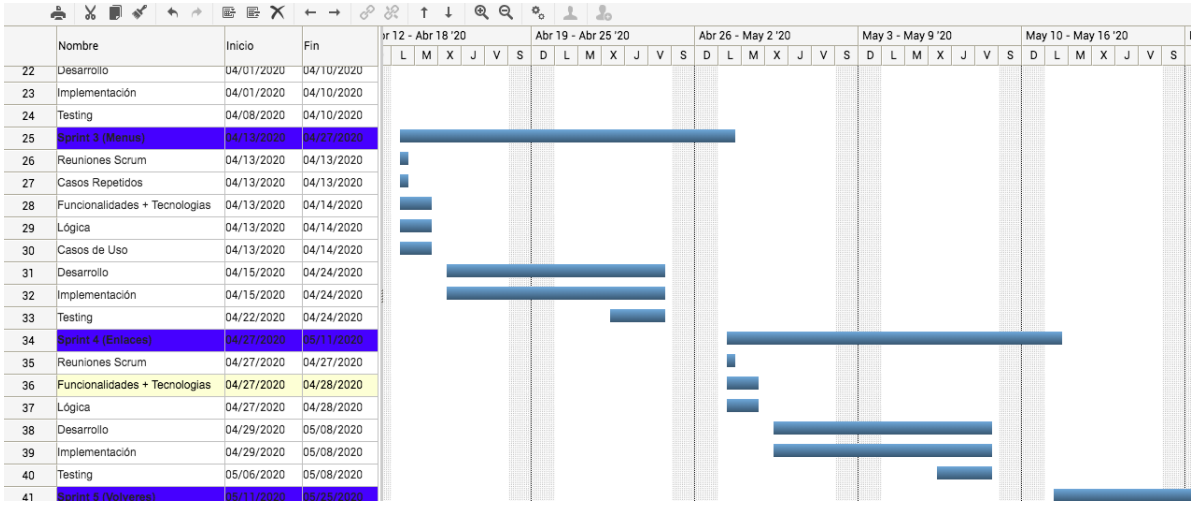
	Nombre	Inicio	Fin
1	Gestión de proyecto	02/17/2020	03/16/2020
2	Contexto y alcance	02/17/2020	02/25/2020
3	Planificación tempora	02/24/2020	03/02/2020
4	Presupuestos	03/02/2020	03/09/2020
5	Documento final	03/09/2020	03/16/2020

Figura 2: Gantt 1



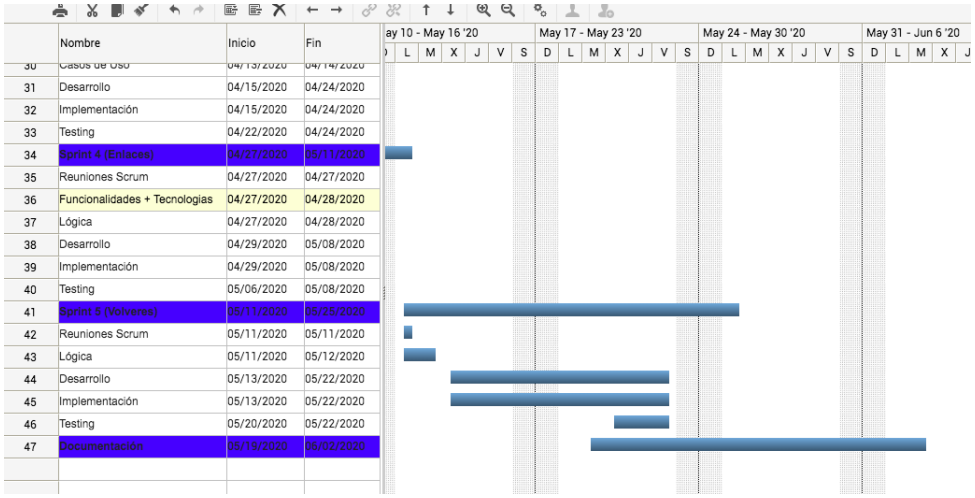
	Nombre	Inicio	Fin
4	Presupuestos	03/02/2020	03/09/2020
5	Documento final	03/09/2020	03/16/2020
6	Sprint 1 (Landings)	03/16/2020	03/30/2020
7	Casos Repetidos	03/16/2020	03/17/2020
8	Nuevas Prestaciones	03/16/2020	03/17/2020
9	Funcionalidades + Tecnologias	03/16/2020	03/17/2020
10	Lógica	03/16/2020	03/17/2020
11	Casos de Uso	03/16/2020	03/17/2020
12	Desarrollo	03/18/2020	03/27/2020
13	Implementación	03/18/2020	03/27/2020
14	Testing	03/23/2020	03/27/2020
15	Sprint 2 (Landings)	03/30/2020	04/13/2020
16	Reuniones Scrum	03/30/2020	03/30/2020
17	Casos Repetidos	03/30/2020	03/30/2020
18	Nuevas Prestaciones	03/30/2020	03/30/2020
19	Funcionalidades + Tecnologias	03/30/2020	03/31/2020
20	Lógica	03/30/2020	03/31/2020
21	Casos de Uso	03/30/2020	03/31/2020
22	Desarrollo	04/01/2020	04/10/2020
23	Implementación	04/01/2020	04/10/2020
24	Testing	04/08/2020	04/10/2020
25	Sprint 3 (Menus)	04/13/2020	04/27/2020

Figura 3: Gantt 2



	Nombre	Inicio	Fin
22	Desarrollo	04/01/2020	04/10/2020
23	Implementación	04/01/2020	04/10/2020
24	Testing	04/08/2020	04/10/2020
25	Sprint 3 (Menus)	04/13/2020	04/27/2020
26	Reuniones Scrum	04/13/2020	04/13/2020
27	Casos Repetidos	04/13/2020	04/13/2020
28	Funcionalidades + Tecnologias	04/13/2020	04/14/2020
29	Lógica	04/13/2020	04/14/2020
30	Casos de Uso	04/13/2020	04/14/2020
31	Desarrollo	04/15/2020	04/24/2020
32	Implementación	04/15/2020	04/24/2020
33	Testing	04/22/2020	04/24/2020
34	Sprint 4 (Enlaces)	04/27/2020	05/11/2020
35	Reuniones Scrum	04/27/2020	04/27/2020
36	Funcionalidades + Tecnologias	04/27/2020	04/28/2020
37	Lógica	04/27/2020	04/28/2020
38	Desarrollo	04/29/2020	05/08/2020
39	Implementación	04/29/2020	05/08/2020
40	Testing	05/06/2020	05/08/2020
41	Sprint 5 (Volveres)	05/11/2020	05/25/2020

Figura 4: Gantt 3



	Nombre	Inicio	Fin
31	Desarrollo	04/15/2020	04/24/2020
32	Implementación	04/15/2020	04/24/2020
33	Testing	04/22/2020	04/24/2020
34	Sprint 4 (Enlaces)	04/27/2020	05/11/2020
35	Reuniones Scrum	04/27/2020	04/27/2020
36	Funcionalidades + Tecnologias	04/27/2020	04/28/2020
37	Lógica	04/27/2020	04/28/2020
38	Desarrollo	04/29/2020	05/08/2020
39	Implementación	04/29/2020	05/08/2020
40	Testing	05/06/2020	05/08/2020
41	Sprint 5 (Volveres)	05/11/2020	05/25/2020
42	Reuniones Scrum	05/11/2020	05/11/2020
43	Lógica	05/11/2020	05/12/2020
44	Desarrollo	05/13/2020	05/22/2020
45	Implementación	05/13/2020	05/22/2020
46	Testing	05/20/2020	05/22/2020
47	Documentación	05/19/2020	06/02/2020

Figura 5: Gantt 4

7. Gestión del Riesgo

7.1. Planes alternativos y Obstáculos

Tal como vimos en los apartados anteriores, en el apartado de obstáculos y riesgos, vamos a dar a conocer algunos impedimentos que pueden aparecer a la hora de realizar este proyecto.

El primero de ellos es la limitación del tiempo entre sprints. Hay una posibilidad de que ocurra esto, debido al corto tiempo entre cada entrega y review, pero como se usa una metodología Scrum, se realizan unos dailies donde el equipo se comunica y comentan inconvenientes o avances favorables. Así se tiene un control de la realidad actual sobre el proyecto. En caso de que no se realice por completo, este conlleva a añadir las tareas en los siguientes bloques de tareas del sprint correspondiente en forma "defect" (significa que ese trabajo se asume como un extra).

El segundo obstáculo es la inexperiencia en el tema de la usabilidad que he ido viendo durante el transcurso del proyecto. No se había previsto este obstáculo al inicio, pero a medida que iba pasando los días se ha ido aumentado la falta de experiencia en cuanto a saber si el producto final es fácil de usar o si es como se lo habían imaginado.

Uno de los errores más comunes en este tipo de proyectos, donde hay más células que trabajan en distintas zonas de este, es la falta de información o la falta de documentación de los servicios creados anteriormente. Esto puede ser un impedimento general, ya que deberás enviar un mail o pedir ayuda para que te lo faciliten. A veces puede ser un impedimento, por lo cual se le asignará una nueva tarea, una muy sencilla, para que avance y no se quede estancado esperando.

Para solucionar todos estos casos, sería conveniente usar como se ha dicho en la primera limitación. Los "defects" ya que es como pedir una extensión del trabajo, pero siempre teniendo en cuenta que se deberán hacer las tareas correspondientes a ese sprint.

8. Presupuesto

En los siguientes apartados identificaré los recursos necesarios para la realización de este proyecto. Por otra parte calcularé el presupuesto que se debe reservar por si en algún caso surge un imprevisto.

8.1. Identificación de los costes

En este apartado se identificarán los gastos principales del proyecto, ya sean recursos humanos, de hardware,..., a su vez se nombraran los usuarios ("los miembros del equipo") que aparecerán en escena y desarrollarán un papel fundamental en cada uno de los sprints. Hay que comentar que este proyecto será realizado por el autor de dicho TFG, por lo que para realizar la estimación se buscará por cada rol sus respectivos sueldos.

A la hora de calcular la amortización, utilizaré la siguiente fórmula:

$$(preciototal \times duracióndelproyectoenhoras) \div Vidaútilenhoras \quad (1)$$

Hay que comentar que a la hora de calcular la amortización, se ha de tener en cuenta que la vida útil es mucho mayor a la duración del proyecto. Es por eso, que a la hora de calcularlo, me basaré en la vida del proyecto, no del dispositivo en sí.

8.1.1. Recursos Humanos

Los principales roles que se darán son:

- Project manager (PM) → 12,30 €/hora
- Analista (A) → 8,52 €/hora
- Desarrollador/programador (D) → 8,97 €/hora
- Tester (T) → 7,63 €/hora

Hay que comentar que el precio por hora es en base al sueldo que me ha proporcionado la empresa donde estoy trabajando, Indra. A parte, cada uno hace referencia a un empleado junior, y decidimos utilizar este tipo de perfil para realizar todos los cálculos.

Para determinar el precio de cada uno de los miembros del equipo, lo que necesitaremos serán las horas en las cuales este miembro ejercerá sus funciones respectivas. Por ello usando la tabla que anteriormente he explicado, en el apartado de la descripción de tareas, tenemos lo siguiente:

Código	Nombre	Tiempo	Precio	Rol
T0	Gestión de proyecto	63 h	774,9 €	PM
T0.1	Contexto y alcance	25 h	307,5 €	
T0.2	Planificación temporal	9 h	110,7 €	
T0.3	Presupuestos	10 h	123 €	
T0.4	Documento final	19 h	233,7 €	
T1	Análisis	150 h	1278 €	A
T1.1	Casos repetidos	5 h	42,6 €	
T1.2	Nuevas Prestaciones	5 h	42,6 €	
T1.3	Funcionalidades + Tecnologías	50 h	426 €	
T1.4	Lógica	60 h	511,2 €	
T1.5	Casos de Uso	30 h	255,6 €	
T2	Desarrollo e implementación	290 h	2601,3 €	D
T2.1	Desarrollo de las funcionalidades	130 h	1166,1 €	
T2.2	Implementación de componentes	130 h	1166,1 €	
T2.3	Nuevos servicios	30 h	269,1 €	
T2.4	Testing	40 h	305,2 €	T
T3	Documentación	100 h	1230 €	PM
T3.1	Memoria	70 h	861 €	
T3.2	Defensa TFG	30 h	369 €	
T4	Reuniones Scrum	18 h	221,4 €	PM
T4.1	Restrospectiva, Planning, Review	18 h	221,4 €	

Tabla 2: Estimación de costes humanos por tareas

Total:	6410,8 €

Tabla 3: Estimación total de costes humanos

Para terminar, a todo esto se le tiene que añadir el precio de la seguridad social, que representa un 30 % del salario bruto, por lo cual, tenemos que el

precio en recursos humanos se incrementa al punto de: 6410,8 * 1,30 = 8334,04 €.

8.1.2. Recursos Hardware

Por lo que respecta a los gastos en dispositivos hardware, se utilizará mediante el usos de portátiles. Como es un aplicativo web, entonces no hará falta el uso de otro dispositivo para las pruebas, sino que se utilizará el mismo a la hora de realizar las pruebas.

Para cada una de las tareas, se utilizará siempre el portátil, tanto en la parte de análisis como la de testing, ya que las unidades a tener en cuenta son equivalentes al número de miembros del equipo.

Nombre	Precio	Unidades	Precio Total	Vida útil	Amortización
HP elitebook 840 g5	1499 €	4	5996 €	6 años	75,406 €

Tabla 4: Estimación total de costes hardware

8.1.3. Recursos Software

En los recursos software, se utilizará Oracle Java Desktop que se pagará únicamente la suscripción durante estos 3 meses, debido a que se paga mensualmente. A parte, el ordenador tendrá integrado un Windows 10 como sistema operativo. Como he dicho anteriormente, el número de unidades para cada componente es equivalente al número de miembros del equipo.

En Windows 10 y Microsoft Office 365 no he rellenado las columnas de Vida útil y Amortización debido a que la vida de estos productos es infinita. Perdurará para siempre, no tiene fecha de deterioro, como en los otros casos.

Nombre	Precio	Unidades	Precio Total	Vida útil	Amortización
Oracle Java	2,5 €/mes	4	40 €	3 meses	12,240 €
Windows 10	48,99 €	4	195,96 €	-	-
Microsoft Office 365	149,99 €	4	599,96 €	-	-

Tabla 5: Estimación total de costes software

Por otra parte comentar que también se han utilizado otras herramientas software pero de manera totalmente gratuitas, como son: atom, un editor de textos, GitLab, para tener un repositorio donde controlar las subidas y almacenaje del código,...

8.1.4. Gastos Generales

Por último hay que comentar todos los costes que no sean reflejados de manera directa, ya sea el uso del portátil, . . . , pero que también se han de tener en cuenta. La electricidad y el agua son aproximados ya que no se podría saber con exactitud.

En lo que respecta a la vida útil del proyecto, únicamente se utilizarán todos estos recursos durante la duración del proyecto en sí, esto quiere decir que luego ya no tendremos más acceso a todos estos. Es por eso que el total hace referencia al presupuesto final, por cada producto, para los 4 meses que estaremos ocupándolos.

Producto	Precio por mes	Precio Total	Vida útil	Amortización
Local	350 €	1400 €	4 meses	316,917 €
Internet	42,99 €	171,96 €	4 meses	38,92656 €
Electricidad	92,98 €	371,92 €	4 meses	84,1914 €
Agua	43,457 €	173,828 €	4 meses	39,3494 €

Tabla 6: Estimación total de costes generales

8.2. Contingencias

Se ha decidido hacer una reserva dependiendo de la probabilidad en la cual podría ocurrir una contingencia dependiendo de que recurso estemos tratando, ya que cada uno es independiente del resto.

Hemos decidido que no sean los mismos porcentajes para cada uno, debido a que la probabilidad de que fallen son distintas para cada uno. Con todo esto tenemos que el presupuesto reservado en caso de contingencias sería el siguiente:

Recurso	Precio Total	Porcentaje de Contingencia	Reserva
Humanos	8334,04 €	5 %	416,702 €
Hardware	5996 €	2 %	119,92 €
Software	835,95 €	2 %	16,719 €
Gastos Generales	10 %	173,828 €	211,7708 €

Tabla 7: Estimación total de costes de contingencias

Después de calcular las contingencias por cada recurso, podemos ver que tenemos un margen (la “reserva de contingencia”) de: 765,1118 €.

8.3. Imprevistos

El principal problema que se podría dar sería, en el caso de tener que utilizar un servicio implementado por otra célula y que no hubiera información suficiente para poder entender los métodos. Por ello, suponiendo que hay un 20 % de posibilidades de que pase, y dependiendo del retraso que se pudiera dar, vamos a suponer que sea de unos 5 días, el coste total sería de unos 71,76 €, teniendo en cuenta que el miembro que se encargará será el desarrollador.

Otro de los imprevistos, sería que fallara el ordenador. En el caso de que esto ocurriera, que tiene una probabilidad mínima, ya que en un principio sería un ordenador nuevo, tendríamos que tener un margen para estos casos que sería de 8 €, teniendo en cuenta que el coste de la reparación fuese de 200 €, para los 4 miembros, y haya una posibilidad del 1 %.

9. Control de gestión

Como lo que hemos estado realizando ha sido un presupuesto de manera estimada, para poder controlar del todo bien las desviaciones reales, lo que haremos será después de cada sprint, donde se realizan unas ciertas tareas, indicar las horas reales de cada una de estas y el miembro que lo haya realizado (también se tendrá en cuenta todos los otros costes, tanto de hardware como de software, incluso los gastos generales). Con ello, tendremos la posibilidad de realizar una comparativa entre las que hemos supuesto y las que hemos efectuado.

A todo ello, se irá actualizando el presupuesto en base a las horas reales consumidas por cada una de las tareas realizadas, y los gastos de imprevistos que hubiera habido durante esta etapa.

Para calcular las desviaciones del proyecto seguiremos estos criterios:

- Coste estimado (sin contar contingencias ni imprevistos)
- Coste real en realizar las tareas

Si la resta entre estos dos costes (coste estimado - coste real) da un resultado negativo, querrá decir que ha habido una desviación/imprevisto, y nos obligará utilizar el presupuesto que teníamos reservado para las contingencias.

10. Arquitectura

10.1. Visión General

Los equipos de desarrollo se dividen por los procesos de negocio (PN) existentes. Con este tipo de organización, conseguimos que cada uno de los equipos se pueda centrar en una sola operativa y que vaya cubriendo todo el desarrollo que hay. Este método sirve solo para aquellos que el negocio esté aislado, esto quiere decir que no necesite de otro PN.

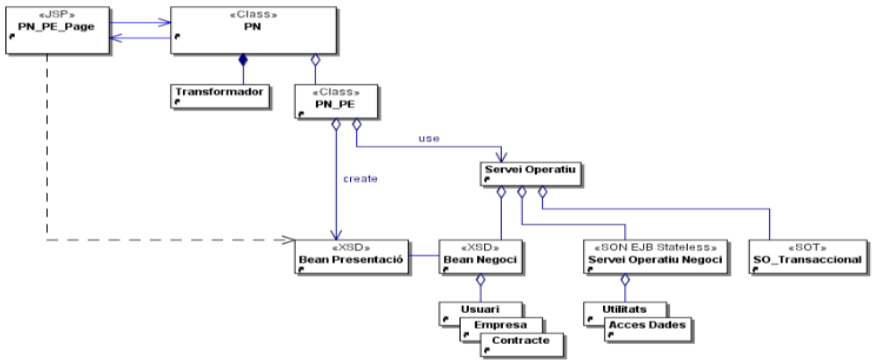


Figura 6: Diagrama de clases

En el diagrama anterior se puede observar lo siguiente:

- Cada página trabaja con su proceso de negocio mediantes peticiones(PE).
- Cada página depende de un Bean Presentación, que es el contenedor que le pasaremos para poder mostrar los datos y las estructuras taglibs.
- Estas implementaciones, se componen de diferentes elementos que atienden a cada petición.

Cuando llega una petición a un PN, se llama a un transformador donde procesa los datos de entrada para su futuro uso, y se crea un Bean de negocio que contiene los datos de la llamada Web. Este objeto creado por cada petición, se deja en una bolsa (librería interna de la empresa) donde se almacena para que el PN pueda recibir los beans y así pasar directamente a cada uno de los servicios operativos que los necesite. Toda la implementación que hay en los servicios operativos se modela como un Entreprise JavaBeans statless (EJB).

Su objetivo es dotar a los programadores de un modelo que les permita abstraer de los problemas generales (sobre la persistencia, las transacciones,...) para que se pueda centrar en otros aspectos, como el desarrollo de la lógica de negocio. Acto seguido se guardará toda la información en un bean de presentación que se pasará a la página, para poder mostrar el layout y se generará una bolsa de salida para la siguiente petición.

En la siguiente figura se muestra un diagrama de secuencia donde se ve más claro lo comentado.

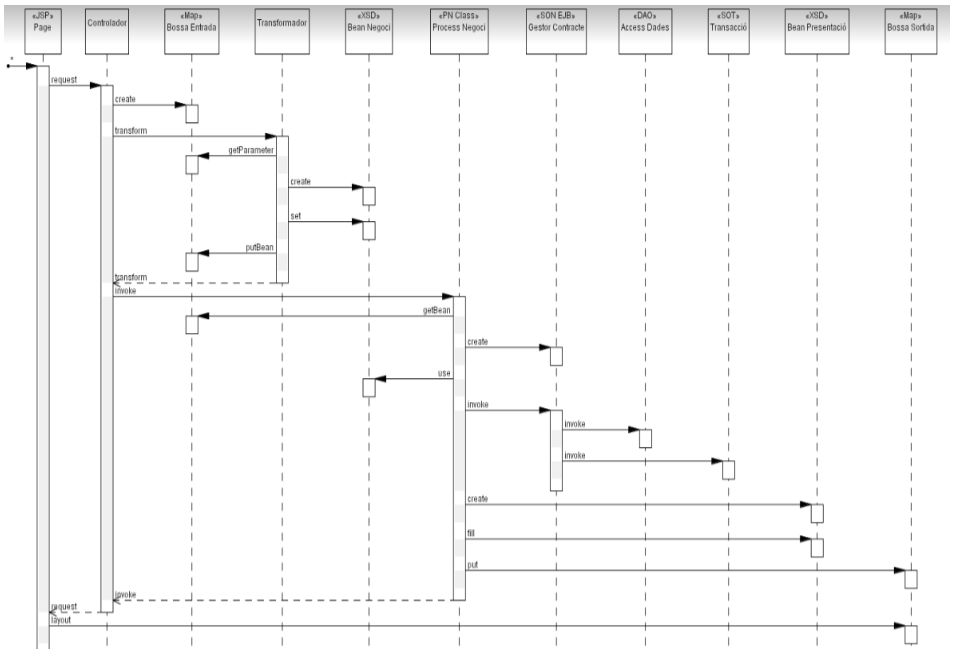


Figura 7: Diagrama de secuencia

10.2. Proceso Negocio (PN)

Este componente encapsula una agrupación de operaciones de negocio. Contiene la tanto modelización entre las capas de presentación y negocio, como las restricciones de seguridad y las validaciones funcionales (pre/post condiciones, gestión del flujo,...). En el trasfondo se tendrá en cuenta diferentes aspectos.

10.2.1. Flujo de navegación y estado

El flujo de navegación esta relacionado con los estados que puedan haber. Aún teniendo en cuenta que cada petición comporta un cambio del estado, el diagrama de flujo no tiene que coincidir completamente con el de estado. Por ejemplo, una petición del usuario puede lanzar una petición de trabajo asíncrono, que realizarse aparte de la intervención del usuario, puede tener sus propios estados.

Por otro lado, de una manera general, un estado equivaldrá a una petición. Todas estas se etiquetarán mediante un campo llamado PE (PEtición).

Dado que el proceso de negocio es responsable de controlar el flujo de navegación funcional entre peticiones, es imprescindible que esté definida, previamente, toda la navegación.

10.2.2. JavaBean

Antes de empezar la construcción de todo, hay que tener claro cuales van a ser las interfaces. Esto conlleva a definir cada JavaBean con los cuales se tienen que trabajar. Se da por entendido que un Javabean es una clase de implementación limpia (sin las dependencias de librerías) que se utilizarán como contenedores de datos y no se usará ninguna lógica.

10.2.3. Configuración y fichero de Literales

Por cada uno de estos PN hay que crear dos tipos de ficheros: uno de configuración y otro de Literales, ambos son Java properties.

Todos estos properties, se identificarán con un atributo llamado versión, que se tendrá que ir actualizando y siempre manteniendo la compatibilidad. Entre todos los datos que se tienen que configurar, se encuentran: aquellos que hacen referencia a la seguridad, al nivel de acceso, los identificadores, el campo de que identifica el estado de petición,... El formato que tendrá es el siguiente: *property_name = Value property.*

```
ID=GUS
version=3
*.*.*.activo=1
*.*.*.nivelacceso=0

*.*.*.1.activopeticion=1
```

```
*.*.*.1.nivelaccesopeticion=3
```

Listing 1: Fichero de configuración

En este pequeño fragmento podemos observar un ejemplo de lo que contiene nuestro fichero properties, usados por la arquitectura. El id, representa el nombre que identifica de manera única un fichero (asociado a un PN). La versión, representa el valor numérico que indica la versión de la especificación usada para definir los parámetros de configuración. Por otro lado el activo es el valor binario "0.º "1"que indica si el PN esta activo en su totalidad o no, esta propiedad permite activar y desactivar un proceso de negocio ya instalado. El nivelacceso es el valor numérico de rango 0-3 que indica el nivel de seguridad requerido para desempeñar el PN. Entendemos por activopeticion cuando contiene un valor binario "0.º "1"que indica si una petición es activa; El valor 0 indica que no esta activo i el valor 1 que si que lo esta. Para acabar el nivelaccesopeticion es un valor numérico de rango 0-3 que indica el nivel de acceso de seguridad requerido para poder ejecutar una determinada petición.

En cuanto a la configuración de literales por cada proceso de negocio se ha de crear un fichero de literales de tipo Java de propiedades. El fichero de literales tiene el siguiente formato:

"Nombre de la propiedad- "Valor de la propiedad"

```
version=2
ID=elogusc000101.prop
*.*.01.004=Identificador
*.*.01.005=Carpeta
```

Listing 2: Fichero de Literales

Observando el extracto anterior podemos ver como es el inicio del fichero de literales. En este caso el id es distinto al que se ha desarrollado antes, ya que en este caso solamente es el nombre que identifica de manera única el fichero. En este caso la versión es completamente igual a la que se emplea en el fichero de configuraciones. En los siguientes casos podemos apreciar que el 01 hace referencia al idioma, en este caso 01 sería la lengua española, por otro lado el 004 o el 005 representa al nombre literal, el cual hace referencia en las paginas que lo necesitan.

10.3. Peticiones (PE)

Un PN puede llamar a varias peticiones, al margen de esto, en cada pantalla se suele identificar una única petición. Toda la lógica que contiene esa misma pantalla se centraliza por código y se accede mediante una petición asociada.

Sus funcionalidades están compuestas por las siguientes:

- Recepción de datos del cliente i de sesión de los usuarios.
- Validación de la información recibida tanto sintáctica como semánticamente.
- Preparación de la información antes de invocar al negocio, ya que este es reutilizable y en según que contexto se puede dar el caso de que sea posible de reutilizar.
- Hacer una llamada a los servicios operativos pertinentes.
- Control de decisión, se recoge la salida de cada servicio operativo y responder en consecuencia.
- Crear un control de flujo, enviar al cliente los datos o el mensaje de error que corresponda.

Conociendo el PN, la petición y el estado podremos saber a que jsp nos redirige la operación.

10.4. Servicio Operativo (SO)

Un servicio de negocio representa un proceso que realiza una operación de negocio. Por eso será necesario tener de antemano que operaciones son necesarias y que tipos de SO involucran. Los motivos de su implementación se resumen con la necesidad de hacer una transacción con back-end: definir la operación a invocar, valores de retorno,...

Está muy enlazado con los JavaBeans donde se reflejan los datos que deben contener. Al ser llamadas por varios PN, su objetivo es que sea reutilizable, por lo cual tiene que estar libre de cualquier dependencia sobre el PN por el cual, inicialmente, se habría desarrollado.

En resumen, se podría decir que un SO se centra únicamente en ser un proveedor de datos y de operaciones de negocio.

Existen dos tipos de SO: los servicios de negocio y los transaccionales.

10.4.1. Servicio Operativo de Negocio (SON)

Los servicios operativos de tipo no transaccional aglutinan las utilidades y los componentes de acceso a la base de datos (DAO). Dado la gran disponibilidad de componentes disponibles, cada uno implementa su propia interfaz de trabajo. El único factor común necesario en todas estas es que no hagan referencia a ningún JavaBean de ningún proyecto en particular.

Por lo tanto, tan solo se dedica a gestionar el proceso de negocio, utilizando, si es necesario los servicios prestados por otros SO's, de utilidad o transaccionales. Un ejemplo de esto se puede ver en el siguiente extracto donde un SON, en este caso SON_GUS, utilizará la llamada de otro SO's:

```
private SON_04086 lnkSON_04086;

...

this.lnkSON_04086 = new SON_04086();
this.lnkSON_04086.setLlistaPerfils(this.llistaPerfils);
this.lnkSON_04086.setUsuariContracte
        (this.getUsuariContracte());
this.lnkSON_04086.setCOMONL(this.getCOMONL());
this.lnkSON_04086.setNUMELE(this.getNUMELE());
this.lnkSON_04086.setOPCION(this.getOPCION());

wEstado = this.lnkSON_04086.ejecutar(cs,logRegPres);

this.llistaPerfils = this.lnkSON_04086.getLlistaPerfils();
...
this.identificador = this.lnkSON_04086.getIdentificador();
```

Listing 3: Código SON

Todas las ejecuciones de los SO's devolverán un estado, una devolución numérica, con el valor de la operación ejecutada. Se podría seguir unos estados ya predefinidos, como por ejemplo la lista de códigos de estado de respuestas de http, pero decidieron usar uno propio donde si es mayor o igual que 10 significará un error, explicando el motivo de este.

10.4.2. Servicio Operativo Transaccional (SOT)

Se utilizan este tipo de servicios para realizar una transacción, habitualmente con HOST (donde se encuentra la API de mensajería).

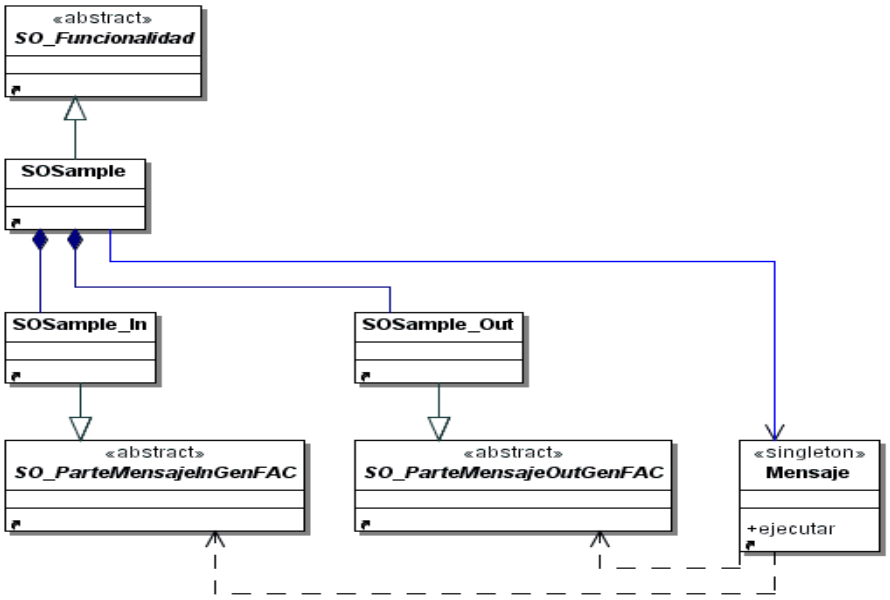


Figura 8: Diagrama de clases SOT

Además viendo la figura 8 podemos observar que el componente SOFuncionalidad es responsable de llamar a la API que ejecuta la operación correspondiente. Esta clase depende directamente de las clases ParteMensajeIn y ParteMensajeOut que se heredan.

- **SO_Funcionalidad**: se encarga de implementar funcionalidades como llenar la mensajería de entrada, tratar la salida de la mensajería, incluido los casos de error,...
- **ParteMensaje In**: es el componente que modela la entrada de datos según la transacción a ejecutar. Su responsabilidad recae en crear el contenedor de los datos, pasar los datos a la capa de transporte.
- **ParteMensaje Out**: es el componente que modela la salida de datos según la recogida de la transacción a ejecutar. Su responsabilidad recae en recuperar los datos del contenedor que hemos recibido.

11. Descripción e Implementación

11.1. Casos de usos

En esta sección explicaré las acciones/actividades que están permitidas por cada una de las entidades que aparecen en el total de nuestro proceso, y cómo deberían interactuar con el sistema. En la aplicación en la cual nos encontramos, podemos diferenciar varios actores que están implicados. Dependiendo de cual sea su rol, describiremos sólo el comportamiento externo y no nos pondremos a mirar con detalle la funcionalidad interna, sin hacer referencias a elementos concretos de la interfaz de los usuarios.

Primero de todo explicaré los actores que hay actualmente. Se pueden dividir en dos relaciones, los apoderados y los autorizados. Los primeros son aquellos que pueden realizar cualquier tipo de gestión sobre los usuarios de un mismo contrato, se podría decir que tiene el mismo poder que el titular/los titulares del contrato. Los contratos agrupan a todos los usuarios que tienen algo relacionado con el titular de esta, dicho de alguna manera son los responsables. Sin embargo, a los usuarios apoderados no se les puede realizar la mayoría de las operaciones, únicamente la de consultar y renovación en el caso de que este a punto caducar o ya esté caducado.

Los segundos, tienen menos poder de responsabilidades, dicho de alguna manera, que los apoderados o los titulares. Es por este motivo que únicamente pueden ser consultores o preparadores. Cada una de estas dos relaciones se pueden subdividir en distintas categorías que pueden ser: consultores, preparador y detallados. En resumen, tenemos 7 tipos de actores distintos:

- Consultor Apoderado
- Preparador Apoderado
- Detallado Apoderado
- Todo Permitido Apoderado
- Consultor Autorizado
- Preparador Autorizado
- Detallado Autorizado

Los consultores son aquellos que su nivel solamente les permite lanzar operativas dentro de la aplicación que son de solo consulta.

Los preparadores son aquellos que su nivel solamente les permite preparar operaciones para que luego un apoderado las firme y por consiguiente se ejecuten.

Previamente, para acceder a la parte en la que estamos desarrollando, hay que tener un nivel mínimo de acceso, ya implementado. Exactamente, todos los que pueden acceder son aquellos que no sean consultores o apoderados, esto no quiere decir que los que tengan acceso puedan ejecutar todas las operaciones que comentaré más adelante. Pero tienen acceso a las demás operaciones de la aplicación global. Si se accede a ella con un usuario que no tuviera las acreditaciones necesarias, saldría un mensaje de error que explicará el problema del acceso denegado obtenido.

Una vez dentro, las operaciones que se pueden realizar son las siguientes:

- Dar de alta usuario
- Consulta
- Baja usuario
- Modificación de la vigencia
- Modificación de los permisos
- Renovación de usuario

En el alta de usuario se efectuarán nuevos ingresos de usuarios, con el estado pendiente de activación, siempre con el DNI de ese, donde se les podrá asignar el permiso o el rol (consultor, apoderado,...) que va a tener en un principio. Por lo que respecta a la relación que tenga en el contrato (apoderado o autorizado) este ya estará vinculado al DNI en cuestión y no se podrá reemplazar por medio de la aplicación. Por lo cual, si un DNI tiene asignado que es autorizado, cuando se hace una alta de ese, siempre tendrán los mismos roles posibles de los autorizados, en este caso: consultor, preparador o detallado.

La consulta no es más que obtener los datos principales del usuario del cual se ha seleccionado.

La baja de usuario es una operación de eliminación del usuario en cuestión, sin importar el estado del usuario, sea activo, pendiente o caducado (los tres posibles estados de los usuarios).

En los términos de la modificación, existen dos posibles alternativas, las cuales al realizar una de las dos no excluirá a la otra, ya sea sobre la vigencia de este o de los permisos que tenga. Hay una condición que se debe cumplir en ejecutar esta, y es que si no tiene un estado activado, no se podrán modificar los permisos. Aparecerá un mensaje de error explicando el motivo de este estado.

Por último, la renovación de usuario aparecerá únicamente en aquellos que su estado sea caducado o que su vigencia este a punto de vencer en los siguientes

30 días (hay que activar al usuario, sino aparecerá como pendiente de activar). Una vez dentro, lo que nos permite realizar es renovar el contrato realizado al usuario. Se podría considerar que es parecido a la modificación de vigencia pero que solo aparecerá cuando se cumpla el estado o la fecha de vencimiento.

Hay que tener en cuenta que las operaciones consultar y renovación del usuario, siempre y cuando se pueda, estarán disponibles para realizar sobre cualquier usuario, pero las demás operaciones estarán solo disponibles en los usuario apoderados. Cuando me refiero a la disponibilidad, viendo la figuras siguientes (figuras 12, 13 y 14), significa que aparecerán en el drop-dwown de la derecha, en las acciones sobre cada usuario, las cuales se podrían ejecutar sobre ellos.

```
if (!AutorizacionOperativaRiesgo.
                    Validar(caixaSession, entrada)) {

    sortida.put(ConsGUS.TORNA_PN,
        ConsGUS.PROCESO_NEGOCIO);

    sortida.put(ConsGUS.TORNA_PE,
        "" + ConsGUS.PET_PREVI_CONFIRMACIO);

        sortida.put(ConsGUS.RUTA,
            ""+ConsGUS.PET_CONFIRMACIO_ALTA_USUARI);
        sortida.put("ERROR1", "001");

        sortida.put("TITULOERROR",
            ConsGUS.LIT_TITOL_ERROR_ALTA_USUARI);
        return NOK;
}
```

Listing 4: Código de comprobación de la operación

Con esta comprobación miramos si con el usuario actual podemos finalizar dicha operación. Pero en este caso, utilizamos una clase que no ha estado hecha por nosotros por el cual a la hora de hacer el testing, no tenemos que probarlo porque ya debería estar comprobado por el equipo encargado.

11.2. Página Inicial - Administración de usuarios

Al iniciar este proyecto, lo primero que debía hacer era inspeccionar la página inicial, la vista que todos los usuarios verán primero. Lo primero que observe fue que parecía antigua y un poco triste en lo que respecta al diseño. Es una simple

tabla donde se hace una paginación de todos los usuarios existentes vinculados a la empresa, aunque se dividen en distintos roles que pueden realizar cada uno de ellos. Este pequeño detalle hace que sea un poco complicado buscar en el momento uno de estos por los roles, o que se visualizen mejor todos los que hay disponibles, ya que únicamente están en esa tabla apilados sin seguir ninguna orden.

A parte, el primer enlace que se puede observar en la parte superior, nos redirige a otra página donde hay más usuarios (de otros tipos). Vemos que con esta manera de separar los distintos miembros dificulta a la hora de la búsqueda de estos y que si nadie se lo comunica, no sabrán que en esa vista dispondrán de otros usuarios que no encontrarán en la tabla inicial.

Otro caso, es la selección de las operaciones que quieras realizar sobre un usuario. Actualmente podemos observar que para determinar qué acciones queremos ejecutar sobre este, debemos seleccionarlo y acto seguido ir al final de la página a escogerlo. No es algo muy directo y eficaz, ya que al seleccionar se debería mostrar las opciones pertinentes de cada uno. Por otra parte, no se muestran qué opciones se puede ejecutar por cada uno de la tabla, debido a que no todos pueden hacer las operaciones que se muestran abajo de la imagen. Hay que tener en cuenta varios aspectos que más adelante comentaré, ya que estará relacionado con los casos de usos. Además, hay que comentar que la opción de alta de un nuevo usuario debajo del todo, no es un buen lugar para depositarlo, debido a que es una funcionalidad esencial que el administrador pueda encontrar con respectiva facilidad.

Por último pero no menos importante, es necesario un buscador interno. No se llega a percibir esta necesidad hasta el momento que veas que tienes unos centenares de usuario, no me imagino si son miles, bajo tu "responsabilidad" y tienes que ir buscándolo en la tabla. Este problema va relacionado con el primero que he explicado más arriba, ya que iría relacionado con la tabla y la gran multitud que miembros que habrá. Es por ello, que esto debería ser importante en nuestra futura página. Puede convertirse en el mejor aliado de nuestro cliente donde se podría ahorrar un valioso tiempo.

Todo esto, se puede deducir de primera vista, sin tener en cuenta el código que hay detrás y cómo se ha solventando todo. La parte del análisis también tiene en consideración esta parte por el motivo que debemos mejorar lo mejor que podamos y no malgastar ningún recurso más, como llamadas innecesarias o ineficiencia en el código y tener que sobrecargar los servidores para usos indebidos.

Podemos observar todo lo que he comentando anteriormente en la siguiente imagen (Figura 9). Cabe recordar que todos los datos que iré exponiendo son datos ficticios, ninguno valor es real debido a la confidencialidad con la empresa cliente.

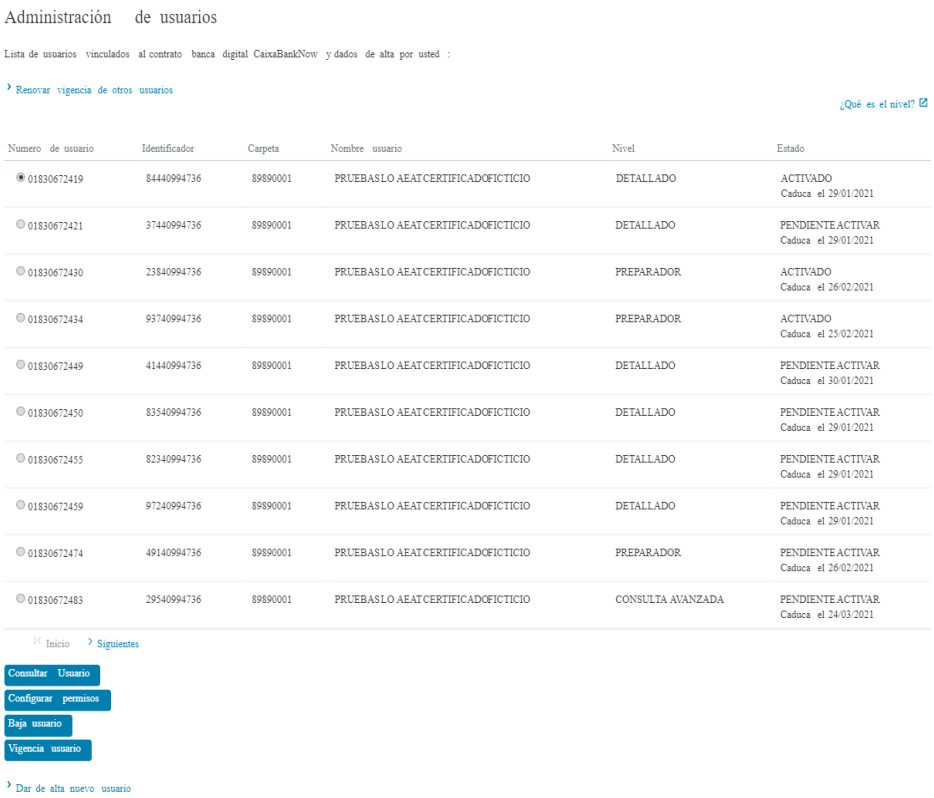


Figura 9: GUS - Antiguo administración de usuario

Para realizar nuestro trabajo necesitamos saber una series de puntos importantes a la hora de implementar la lógica de negocio y los casos de usos de nuestro aplicativo.

Por un lado, existen dos clases de usuarios los cuales se pueden dividir entre los diferentes miembros de la empresa (dicho de otra manera, qué tipo de relación tienen), los apoderados y los autorizados. Por defecto, cada uno de estas clases tienen sus propias características y funciones que puedan efectuar o no sobre los demás personas del mismo contrato, de Línea Abierta de la Caixa. Por ese motivo es recomendable encontrar qué relación tienen los usuarios y separarlos, ya que anteriormente no se tenían en cuenta y se añadía directamente en la tabla mostrada arriba (figura 9). Continuando en esta línea, cada uno de estos, les separa un nivel que se les añade a la hora de darse de alta tanto de manera online o directamente desde la oficinas del banco. Pero teniendo siempre en cuenta el tipo de relación que tenga, ya que si eres de uno como del otro, no comparte por completo el nombre de niveles que existen.

Por otra parte, nos piden que les interesaría tener en cuenta las funcionalidades más destacadas de los usuarios a la hora de realizar cualquier búsqueda u operación. Con esto lo que se busca es tener como un contador o tener anotado las operativas que más usen de manera diaria.

Para ello la solución más eficiente que hay, sería utilizar un gestor de etiquetas, llamada TMS (Tag Management System) ya que en un principio nos enfrentamos con un aplicativo que tendrá un elevado número de visitas en las distintas páginas. Un resumen de su funcionalidad es la de enviar información proveniente de la página o de los enlaces de esta a una central de medios, o en nuestro caso una solución de web analytics. Con esto, lo que se permite es medir la audiencia o la llamada tasa de conversación, que no es más que controlar el tráfico que concurre en esa página. Esto también ahorra trabajo ya que si en toda la operativa utilizamos el mismo sistema de tags y no dependeremos de que integren nuestros tags, si es el caso de que generemos distintos por cada página.

El TMS que utilizaremos será ApiTealium, que es parecida al Tealium, pero únicamente disponible por la empresa bancaria. Este nos proporciona una recogida de información a través de distintas fuentes, tal como cookies, por los elementos meta,... (caracterizado por ser un TMS). Pero esta apuesta nos proporciona unos templates ya predefinidos, donde facilitan la implementación para una gran cantidad de empresas y nos permite un mayor flexibilidad a la hora de identificar los elementos. Además, actualmente contempla 10000 tags diferentes, lo que significa un menor margen de error en las configuraciones.

Al analizar el código anterior se pudo observar que el paso de información entre las vistas y las diferentes operaciones los datos no están protegidas y desde los logs del servidor se podían visualizar toda la información que se iban traspasando entre las diferentes peticiones. Por lo cual se puede obtener la información sensible de todos los usuarios, tal como número de cuentas, tipo de contratos,... relacionados con la empresa a la que esté vinculada. Para ello utilizaremos unas librerías de la empresa que lo único que hacen es guardar la información en un contenedor de datos seguros. Esas librerías se llaman RefVals. Cada vez que se crea un valor de este tipo, donde devolverá una cadena hexadecimal, para poder acceder al valor encapsulado.

Para ello lo que se tendrá que realizar será la siguiente. Hay cuatro parámetros que se deben rellenar: el primer valor es el nombre de la variable que queremos guardar. En caso de que ya existiera un objeto con el mismo nombre, se sobrescribiría. El segundo valor hace referencia al valor que queremos almacenar en el contenedor (en caso siguiente sería la variable cuenta) y los dos últimos parámetros hacen referencia al PN y al PE correspondiente donde se inserte el contenedor. En el siguiente extracto podemos ver un ejemplo de como utilizarlo con las cuentas de los usuarios. El resultado de esta llamada sería una cadena hexadecimal, que será necesario a la hora de consultarlo.

```
String ref = caixaSession.setElementoRefVal
    ("CuentaSecreta",cuenta,    GUS    ,"1");
```

Listing 5: Código para crear un RefVal

Para recoger el parámetro que hay dentro del contenedor, hay que llamar al método de getElementoRefVal donde se tienen que pasar 4 parámetros. El primero sería el nombre de la variable que querremos consultar, debería ser igual que el nombre con el que se guardó. El segundo es el hexadecimal generado al insertar el valor a un contenedor refval, se tendrá que pasar por la bolsa de la sesión. Los dos últimos parámetros son los PNs y PEs donde se han asociado el refval previamente insertado.

```
String cuentaSecreta =
    caixaSession.getElementoRefVal
    ("CuentaSecreta", ref,"GUS", "1");
```

Listing 6: Código recoger el contenido de un RefVal

Con esto solucionaríamos este problema que se detecto en los pasos de parámetros en los flujos de las diferentes operaciones.

Otro aspecto a tener en cuenta es que actualmente se utilizan una serie de librerías, llamadas Taglibs, desarrolladas para ser integradas y utilizadas en las páginas JSP. Se creará un tag por el desarrollador en el fichero java correspondiente, y en el fichero JSP se ejecutará esa etiqueta, que no es más que una serie de acciones destinadas a ser utilizadas en la JSP. Esto se considera poco eficiente, ya que se intentará que todas las páginas que sean iguales (por ejemplo, las de confirmación de cualquier operación, resultado de estas mismas,...) sigan los mismos estilos, hablando por la parte estética (márgenes, espaciados,...). Además existe una librería creada por la misma empresa bancaria, donde tienen unos ficheros css, los estilos, marcando las pautas, reglas, separaciones,... utilizadas por todos las operativas. Esto se hace así ya que si es se decide modificar los estilos de un título, por ejemplo, se modifiquen en todas las operativas de la empresa y no tener que ir cambiando uno a uno, fichero a fichero, cambiando el tamaño o haciéndolo a ojo. Esto nos ahorrará mucho tiempo ya que de una manera es como centralizar todas las características/ reglas de las páginas correspondientes.

Lo que haremos será hacer una mezcla de estas librerías, y las reglas de estilo. Con los Taglibs crearemos todas las estructuras necesarias para la correcta mejora de la página y con las reglas de estilos, el correcto diseño y estructuramiento de la página, y de las etiquetas generadas.

A continuación podemos ver (figura 10) cómo sería la vista definitiva de nuestra mejora respecto a la anterior pantalla. Se puede observar un cambio visual que ofrece una mayor limpieza y claridad. No hay un carga excesiva de información, como se podía contemplar antes, debido a que hay una funcionalidad de consulta sobre el usuario, que se explicará más adelante, que recoge toda la información que aparecía en la tabla central. Podemos utilizar todo ese espacio para que añadir otras utilidades que beneficiaría a los usuarios finales. El objetivo no es eliminar los elementos y tenerlo lo más limpio y claro posible, sino que es tener lo esencial para ayudar a los usuarios. Es por que que lo primero que se debía realizar era la nueva manera de mostrar estos.

Se decidió utilizar un menú en formato desplegable teniendo en cuenta el tipo de nivel que tenían y agruparlos según la relación a la que permanecían, explicado anteriormente, y que nos centraremos más adelante.

Además podemos observar que hay una novedad más, que es necesario para cualquier usuario debido a que los cibernautas de hoy en día, según los estudios, se han acostumbrado a navegar a través de búsquedas, por lo cual hacen especial énfasis a Google. Por otra parte, también ahorra tiempo, ya que no les interesa estar perdiendo su valioso tiempo en estar buscando al usuario correcto. Debemos dar la mayor accesibilidad a los usuarios y es por eso que vamos añadirlo con un tamaño mediano, formando así un cuadro de búsqueda y no tener un icono o cualquier otro elemento de un tamaño inferior.

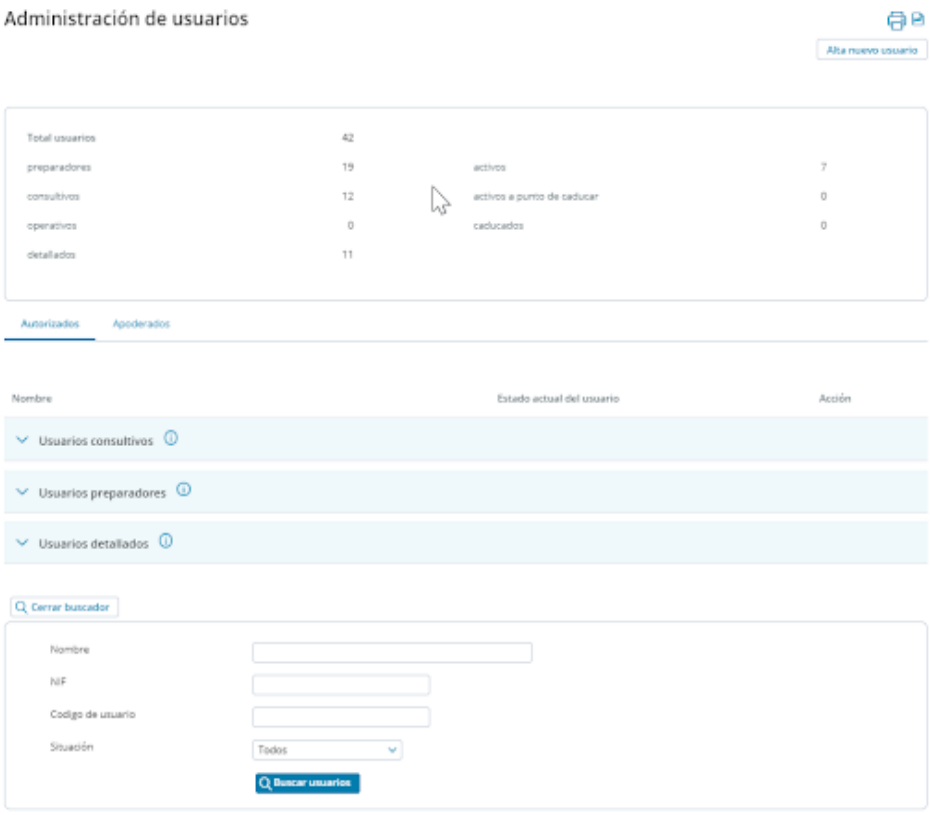


Figura 10: GUS - Nuevo administración de usuario

El primer cambio que podemos considerar es la cabecera de la vista donde se puede observar mejor en la figura siguiente (figura 11). Podemos diferenciar, respecto al a la anterior pantalla (figura 9), que hemos reestructurado todos los enlaces de las operaciones disponibles que habían (modificación, alta usuario,...). En dicha cabecera, hemos habilitado algunas funciones que no se disponían. En este caso, en la parte derecha de la siguiente figura (11), se ven dos nuevos iconos: el de una impresora y el de una pdf. El primero, como bien se puede intuir, es un icono que al clicarlo llama a una función que imprimirá la pantalla en cuestión. El segundo, conseguimos convertir la pantalla en pdf sin tener que salir de nuestra aplicación, y sin usos de interfaces. Es decir, sin tener que ejecutar una ventana o pestaña independiente, por lo tanto no forzaremos a que el usuario salga o abandone nuestra web.

En el uso del icono, teníamos que buscar uno que fuera algo bastante representativo e ilustrativo para la función a la que está asociada. Por esta razón, se decidió escoger un icono simple y sin mucha florituras, pero a la vez que no sea muy anticuado. Por otro lado, al tratarse de un icono universal, no se debe hacer

un rediseño, excepto en todo aquello relacionado en un ajuste de estilo tal como tipo relleno y bordes, colores,..., debido a que podría afectar drásticamente en la usabilidad.

Figura 11: GUS - Cabecera administración de usuario

Además se añade un cuadro con un cálculo de los diferentes tipos de miembros en los que se podrían subcategorizar los usuarios. Con esto también conseguiremos que con todo el caos que había anteriormente, se tenga más o menos un control más preciso sobre las personas a las que tienes bajo tu responsabilidad (esta responsabilidad hace referencia a que un usuario puede realizar cualquier operación, explicadas en los casos de uso, sobre otro en el cual estén vinculados).

Para implementar este parte, se tuvo que cambiar algunos aspectos en la implementación que había existente. Lo que se realizará es una modificación a la hora de hacer la petición a la parte del servidor (back-end). Generamos un listado de usuarios, dependiendo del nivel que tenga cada uno de estos, una vez que nuestra petición nos haya devuelto los usuarios que administra nuestro usuario. Acabada toda esta parte, resultará muy fácil tener el número de usuarios por cada uno de los listados. Aparte, nos ahorraríamos el tener que ir llamando al servicio para recoger los datos por cada uno de los usuarios existentes. Por lo tanto, ahorramos tanto tiempo como por dinero, debido a que por cada llamada se consume un gasto económico, que también tendríamos que tener en cuenta a la hora de hacer la implementación y la mejora en el código.

```
contratosLA = (ContratosLA)listaContratosLA.get(i);

if (!(contratosLA.getDniApo().equalsIgnoreCase
    (lnkSON_GUS.getLnkLlistaUsuaris().getDniTit()))){

    if ("X".equals(contratosLA.getEstadoUsuario())) {
```

```
            wEstado = contratosLA.getEstadoUsuario()
            + " " +
            contratosLA.getFechaCaducidadUsuariosOnLine();

            wActivosApuntoCaducar++;
        } else if ("A".equals(
            contratosLA.getEstadoUsuario())) {
            wEstado = contratosLA.getEstadoUsuario();
            wActivos++;
        } else if ("D".equals(
            contratosLA.getEstadoUsuario())) {
            wEstado = contratosLA.getEstadoUsuario();
            wCaducados++;
        } else {
            wEstado = contratosLA.getEstadoUsuario();
        }
```

Listing 7: Código recoger datos de cabecera

Visto el código, comentaré ciertos aspectos que hay que tener en cuenta. Con la llamada a equalsIgnoreCase() conseguimos que hacer una comparación con otro objeto sin importar las minúsculas y mayúsculas que contenga, esto es provocado por motivos de las letras guardadas en el DNI de los usuarios. Luego de hacer esta comprobación (ya que en el bloque de los usuarios, estará el usuario titular del contrato), hacemos las comparaciones respecto al parámetro que hace referencia a los estados del usuario:

- Si es una 'X', hace referencia aquellos que estén apunto de caducar, pero activo.
- Si es una 'A', hace referencia aquellos que estén activos.
- Si es una 'D', hace referencia aquellos que ya están caducados.
- Por último, hace referencia aquellos que no están activos, es decir, que están pendientes de activación.

Con wEstado vamos a guardar el estado del usuario para la nueva forma de mostrar los usuario. En la figura 12, podemos ver como por cada uno, hay un parámetro 'Estado actual del usuario' donde aparecerá dicha situación del usuario. Pero, si se recibe uno que este apunto de caducar, además añadiremos en el parámetro la fecha próxima de vencimiento.

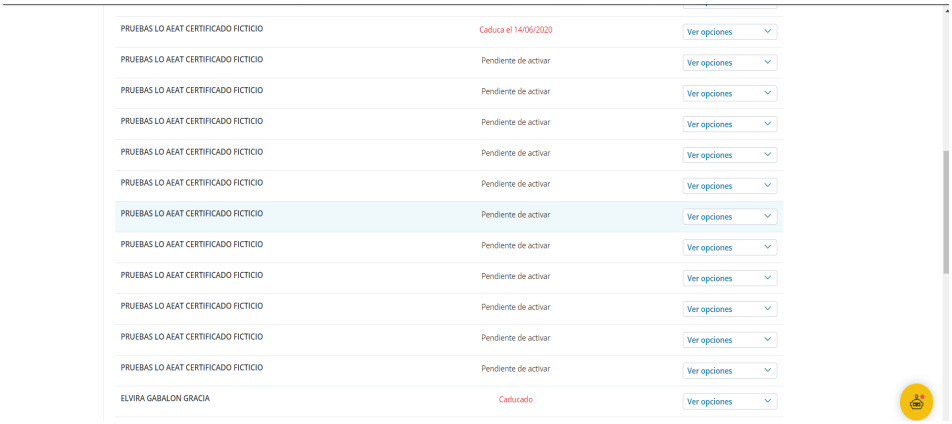


Figura 12: GUS - Estados del usuario

Otro de los cambio realizados sería el nuevo modo de mostrar los usuarios. Como hemos dicho anteriormente, vamos a diferenciarlos entre sus relaciones dentro de la empresa, de esta manera los separamos y conseguimos obtener un orden en la administración y conseguir así que el usuario no se pierda con tantos usuarios bajo su mismo contrato. Algo que se tiene que tener en cuenta es que los usuarios no tienen tiempo para encontrar lo que buscan sino que lo quieren de una manera rápida.

Para su presentación, se decidió implementar menús en forma desplegable. Es así debido a la necesidad de no tener que mostrar una gran cantidad de información y no tener saturada la vista con ello. En la siguiente imagen (figura 13) podemos ver un claro ejemplo de lo dicho. Con esto definido, vamos a agrupar a todos los usuarios dentro de un menú y así tenerlos todos en un mismo lugar. Por ejemplo, imaginémonos que únicamente necesitamos comprobar el estado actual o realizar operaciones en los miembros de nivel detallado, los otros restantes no nos sería de utilidad mostrarlos, por lo cual al tenerlo de esta manera, ocultamos esa información pero con posibilidad de que lo puedan visualizar al desplegarlo de nuevo. Es necesario que estos deban verse de forma clara y deben incitar a hacer clic al usuario, es decir, que éste identifique, que accediendo al mismo, podrá encontrar lo que busca.

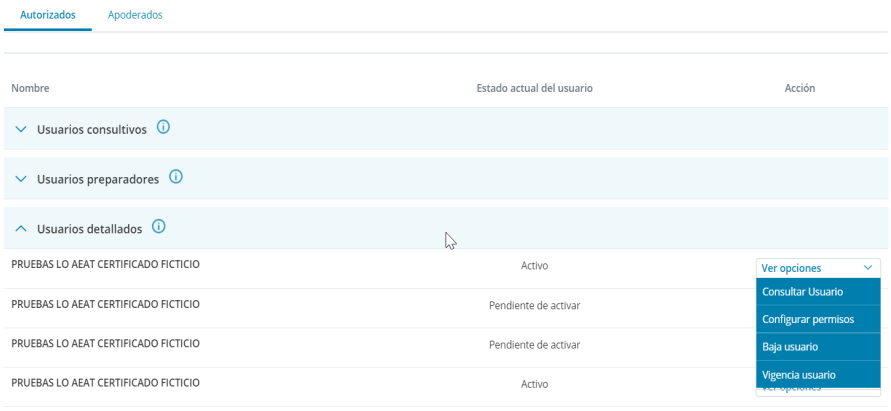


Figura 13: GUS - Tabla administración de usuario

Para ello lo que haremos es utilizar los taglibs, ya que podemos ahorrarnos de ensuciar el fichero jsp con su implementación. Un problema de utilizar los tagligs, debido a que es una etiqueta que generamos para mostrarlo en el fichero jsp, es que ocupa todo lo ancho de la pagina. Por ello, si se obtiene un espacio negativo (los espacios en blanco, separaciones entre los componentes,...) que luego se querrá utilizar, entonces no se podrá añadir. Es por eso que hay que saber cuando utilizar los taglibs o usar html, para no desperdiciar los espacios y así prevenir los posibles disgustos en un futuro, provocados por modificaciones sobre ese componente.

```
<lo:tabla idBean="tablaDesplegableUsuarios1"/>
<lo:paginacion idBean="paginacion01"/>
<lo:tabla idBean="tablaDesplegableUsuarios2"/>
<lo:paginacion idBean="paginacion02"/>
<lo:tabla idBean="tablaDesplegableUsuarios4"/>
<lo:paginacion idBean="paginacion04"/>
```

Listing 8: Código JSP con taglibs

Con este trozo de código conseguimos poder representar la figura 13. Para ello utilizamos los taglibs. Simplificaremos mucho el fichero jsp y así en un futuro podremos volver a utilizarlo para que otra persona lo pueda utilizar en otro lugar, ya que lo tratará como una etiqueta html creado por nosotros

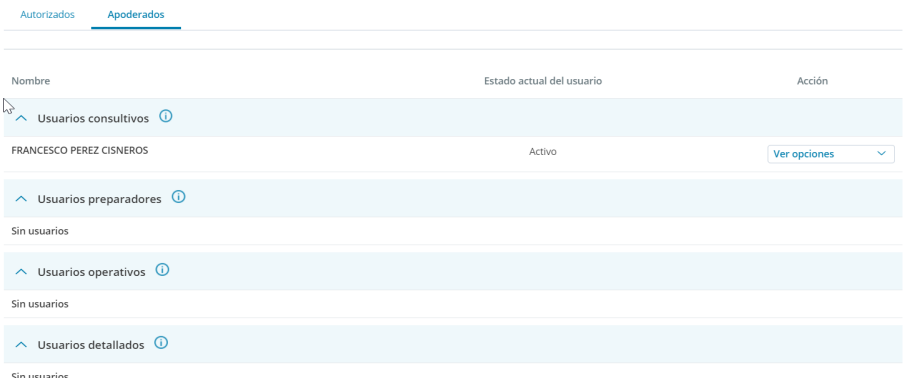


Figura 14: GUS - Tabla Apoderados administración de usuario

Por último en lo que respecta al buscador interno, se hizo de la siguiente manera. Con *lo:form name="DADES"* conseguimos representar una sección que contenga controles interactivos, permitiendo a un usuario interactuar con el servidor web. En java crearemos un objeto llamado DatosEntradaBean, una librería de la empresaria bancaria, donde se podrá rellenar con los inputs del buscador para hacer la búsqueda de los usuarios que hagan match.

```
<lo:divContenedor idBean="divC_VF">
    <lo:form name="DADES">
        <lo:datosEntrada idBean="datosEntrada"/>
    </lo:form>
</lo:divContenedor>
```

Listing 9: Código JSP buscador interno

Continuando con las nuevas incorporaciones, se añadió el gestor de etiquetas para tener un pequeño control sobre la navegación. Por ello, tuvimos que crear nuevas etiquetas para tenerlas almacenadas, ya que no existían.

```
ApiTealium apiTealium = new ApiTealium(this,
        getLiteralTealiumGUS("EVENT_CATEGORY")+":"+
        getLiteralTealiumGUS("EVENT_ACTION")+":"+
        getLiteralTealiumGUS("EVENT_LABEL"));

beanMap.put(BeanFactory.newLiteralBean("apiTealium",
        apiTealium.toString()));

BeanFactory.V2.newLiteralBeanDirecto
    ("javascript:"+ apiTealium.sentEvent
```

```
    (getLiteralTealiumGUS("EVENT_CATEGORY"),
    getLiteralTealiumGUS("EVENT_ACTION"),
    getLiteralTealiumGUS("EVENT_LABEL")) +
    "sendLink();"),

BeanFactory.V2.newLiteralBeanNegocio(LIT_ALTA_USURI),
            BeanFactory.V2
            .newTextoBeanNegocio(LIT_ALTA_USURI));

LineaTextoBean linea_alta =
    BeanFactory.V2.newLineaTextoBean(texto);
beanMap.put("alta_user", linea_alta);
```

Listing 10: Código generación de Tealium

En el fichero de configuraciones se tendrá que añadir estas tres líneas donde estarán los atributos del Tealium que deberán coincidir con los valores de negocio, ya que será la etiqueta que les llegará si alguien clica en el enlace.

```
*.*.*.EVENT_CATEGORY=gestionusuarios
*.*.*.EVENT_ACTION=clic en alta nuevo usuario
*.*.*.EVENT_LABEL=alta usuario
```

Listing 11: Configuración Tags del Tealium

Vamos a implementar dos maneras distintas en lo que respecta al uso de esta tecnología, para su futuro análisis de los datos por parte del equipo de analistas. El código de arriba es un claro ejemplo de la implementación de un evento generado al hacer un clic en un enlace. Con la función toString() conseguimos convertir el objeto apiTealium en una cadena de caracteres donde lo tendremos que insertar al JSP, mediante el beanmap. Esto conllevará a la creación un objeto Apitealium (digitalData) y procesará una función en javascript para su envío correspondiente. Por otra parte, también se puede hacer lo mismo pero al estar dentro de una pantalla y no al hacer un clic en un enlace. En un apartado más adelante (baja de usuario) pondré el ejemplo de este caso.

```
<lo:lit idBean="apiTealium"></lo:lit>
```

Listing 12: Código JSP para activar Tealium

Se tendrá que añadir esta linea en el jsp para que se pueda realizar la funciones de tealium. Se puede comprobar que el id tiene que coincidir con mismo nombre con el que se ha asignado en el java al crear el Tealium.

En la siguiente figura (15) podemos ver, inspeccionando la página, que el botón tiene asociado una función que hará lo que hemos estado explicando.

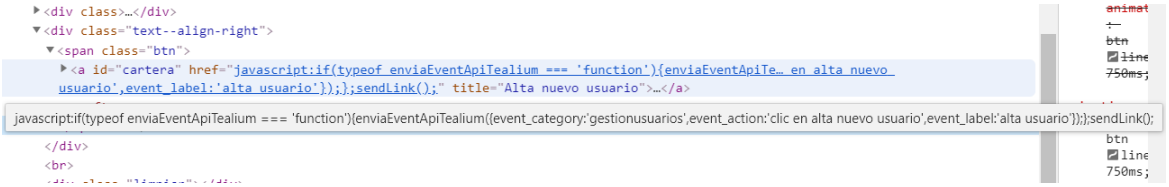


Figura 15: GUS - Función generado por Tealium

11.3. Operaciones

En los siguientes apartados explicaré sobre las diferentes operaciones, únicamente de las que se mejorarán, de las cuales se les pueden ejecutar a los diferentes usuarios.

En el apartado anterior, en los casos de usos, hago referencia a las operaciones en cuestión, pero haciendo una breve introducción de ellos. En este, haré más énfasis sobre las funciones y una explicación, si fuera necesaria, más profunda.

11.3.1. Baja Usuario

En este parte se hablará sobre el flujo completo de la operación de dar de baja a un usuario. Esta se divide en dos partes: la primera se trata de una página de confirmación donde se debe añadir un número de autorización, única para cada usuario, y una segunda página donde se refleja el resultado de la operación, con la información del usuario al que se ha dado de baja.

Por temas de usabilidad, es apropiado que el contenido, dentro de la estructura de datos que vamos a mostrar, esté dividido por cada una de las diferentes secciones en diversos bloques. Viendo la figura 16, podemos observar que los datos del usuario engloba a todo un conjunto de subsecciones en los que se pueden dividir. De esta manera, se facilita una información más ordenada y fácil de identificar para el usuario. Para ello, lo que haremos, es agrupar los datos por subcategorías (bloques) agregando un título destacado para cada uno de estos con el propósito de separarlos y que ayuden de una manera visual a los usuarios. Podemos ver que todos los parámetro son claramente datos del usuario, pero se podría subdividir más en orden de mejorar la información.

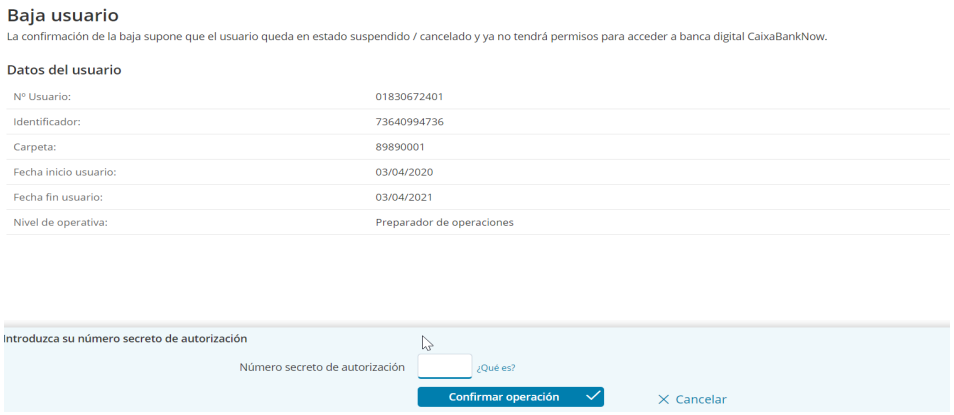


Figura 16: GUS - Baja usuario antiguo

Como se puede ver en los casos de usos, ahora hay que tener un nueva lógica de negocio que debemos implementar, y tener en cuenta en todo momento. Debido a que no todos los usuarios pueden acabar de ejecutar las operaciones correspondientes (se hace referencia aquellas que se tenga que confirmar añadiendo el código de autenticación para finalizar la función, explicada en el apartado de modificación lógica negocio) aunque aparezcan en las opciones, debido a que no hay que ocultar a los usuarios ningún tipo de información. Es por eso que añadiremos una nueva implementación en la cual estaremos comprobando, un nuevo parámetro añadido por la parte del server. Por lo tanto, habrá que modificar la llamada del servicio a la back-end. A la hora de la devolución de los parámetros, hay que recoger ese nuevo y pasarlo a la página en cuestión.

Aparte, se hizo una modificación en la llamada a la Application Programming Interface (API) proporcionando la conexión con la parte del servidor, donde se realizó un cambio en la petición de la query de acceso. Pudimos observar que los datos que se enviaban, al método, no eran necesario para la eliminación de dicho usuario y la consulta a las tablas se ejecutaban de una manera ineficiente. Es por ello que al eliminar los parámetros de filtrado, optimizamos la petición a la base de datos aligerando de manera exponencial los campos a comprobar en cada llamada.

Dicha petición termino siendo tres parámetros a pasar para la ejecución de la baja de usuario.

```
this.lnkSON_04090.setUsuariContracte(Miscelanea.
        tipoStringToString(this.getUsuari()
            .getUsuarioContrato())));
```

```
this.lnkSON_04090.setComonl(Miscelanea
        .tipoStringToString(this.getUsuari()
            .getComonl()));

this.lnkSON_04090.setDninif(Miscelanea.
        tipoStringToString(this.getUsuari()
            .getNifUsuario()));

this.lnkSON_04090.ejecutar(cs,logRegPres);
```

Listing 13: Código nuevo petición baja usuario

Conseguimos reducir el nombre de parámetro a enviar debido a que anteriormente se pasaban más (DataAlta,TiupusCLO y NomUsuari) de los que eran imprescindibles. Para realizarlo tuvimos que hablar con la parte back-end.

Un pequeño problema en esta parte del sprint fue que no hubo comunicación por parte del cliente. Inicialmente, se decidió que fuera a través del front-end quién controlará toda la lógica que había detrás y poder ver quien tenía los privilegios para poder realizar dicha operación sobre el usuario, pero más adelante se decidió que fuera back-end, creando un nuevo parámetro y nosotros modificando la llamada a back recoger el nuevo dato y hacer el control a partir de ella. Por lo tanto, fue una paso hacia atrás y una incidencia que no teníamos en cuenta a la hora de hacer la planificación inicial.

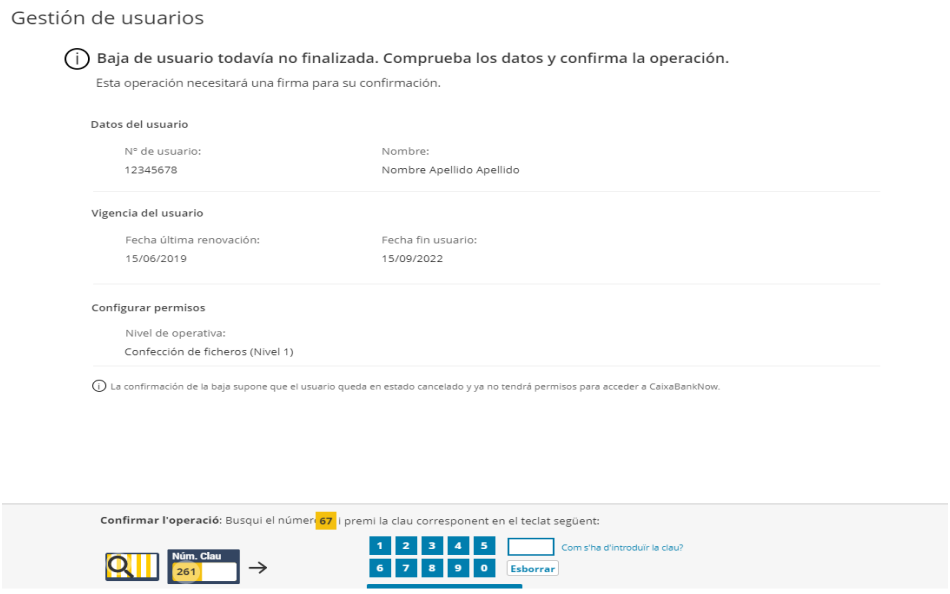


Figura 17: GUS - Baja usuario nuevo

Por lo que podemos observar el la figura 17, la estructura de datos y los componentes de la vista están expuestas de una manera más clara y estructurada. Teniendo una separación razonable (espacio negativo) por cada uno los parámetros a mostrar, conseguimos que el usuario pueda percibir que cada uno de estos campos sean de distintos apartados y no se asemejen a uno solo cómo se percibía anteriormente. Para ello utilizaremos el principio de proximidad del principio de Gestalt. Con ello, permitirá que se entienda de una manera más sencilla la separación de los diferentes bloques de contenido que existen.

En lo que respecta al resultado de la operación, utilizaremos el mismo template que usaremos para generar el muestreo de los datos sobre el usuario. Con ello conseguimos una buena estructura de los datos y la separación de esta.

```
this.lnkSON_GUS=new SON_GUS();
this.lnkSON_GUS.setUsuari(usuari);
this.lnkSON_GUS.executarBaixaUsuari
        (caixaSession,logRegPres);
```

Listing 14: Cógigo operación SON

En este punto, se habrá realizado la llamada a back-end, para la baja de dicho usuario, previamente ejecutada (llamando a la API) con la confirmación del número secreto por lo cual nuestro usuario ya no se mostrará más. Como anteriormente no había el control para realizar la operación nunca daba error, pero actualmente si no puede realizar la operación saltará a una página informando el error que se ha producido. Por lo cual, habrá que añadir dos posibles peticiones resultantes a mostrar dependiendo de quien lo ejecute.

Figura 18: GUS - Resultado baja usuario antiguo

EL SON_GUS no es mas que un DAO (data access obeject), una clase donde se encuentra los parámetros que interesan y luego back-end los rellena. La ventaja de usar el DAO es que cualquier objeto de negocio (aquel que contiene detalles específicos de operación o aplicación) no requiere conocimiento directo del destino final de la información que manipula.

Una de las modificaciones que se pueda apreciar en la siguiente figura (19), de forma distinta respecto a la pantalla de confirmación (figura 17), a parte de la zona de la cabecera, es el botón de imprimir añadido a la parte superior derecha. En la antigua versión no hay posibilidad de imprimir el resultado conforme se había dado de baja, a no ser que se hiciera una captura de pantalla. Para ello se añade una nueva función Javascript proporcionado por esta misma, windows.print() (que hemos comentado con anterioridad).

```
function imprimir(){
    if (window.print) window.print();
}
```

Listing 15: Código JSP función imprimir

Este método es muy útil debido a que es soportable por cualquier tipo de navegador, ya sea Chrome, Firefox o, para aquellos que utilicen Apple, Safari. Por este motivo no era necesario que se implementará una función que lo hiciera o tener que buscar otra librería que nos facilitará este método.

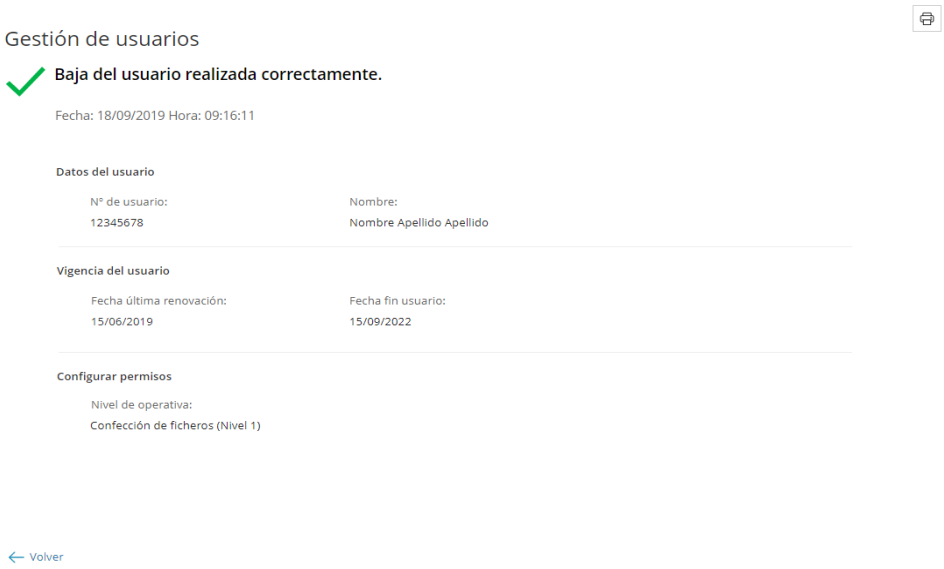


Figura 19: GUS - Resultado baja usuario nuevo

Por otro lado, se añadió una ventana modal (vista en la figura 20) donde se muestra una tabla con las actuales configuraciones de permisos de los usuarios detallados. Únicamente, estos dispondrán de este ventanal, a diferencia de los demás (como se puede visualizar en las figuras de arriba 16-19), debido a que con este tipo de usuarios se les pueden hacer múltiples combinaciones de permisos disponibles para las distintas acciones, ya que no solo es de esta operativa sino de las demás. Por lo que respecta a los otros usuarios, no sería necesario puesto que ya están prefijados las acreditaciones que tendrían en la aplicación. Para ello, como anteriormente no había esta opción, durante la recogida (llamada a back-end) de información en la primera pantalla pediremos que nos devuelvan en la respuesta un array, solo de aquellos que cumplan con este perfil, con sus acreditaciones que mostraremos en el pop-up. Con esto, conseguiremos que con una sola llamada a la parte del servidor, ahorraremos múltiples llamadas innecesarias mientras que en una sola, lo tendremos almacenado en el storage de la web (llamada CaixaSession). Esto solo persistirá durante la sesión en la que se haya conectado o mientras esté abierto el navegador.

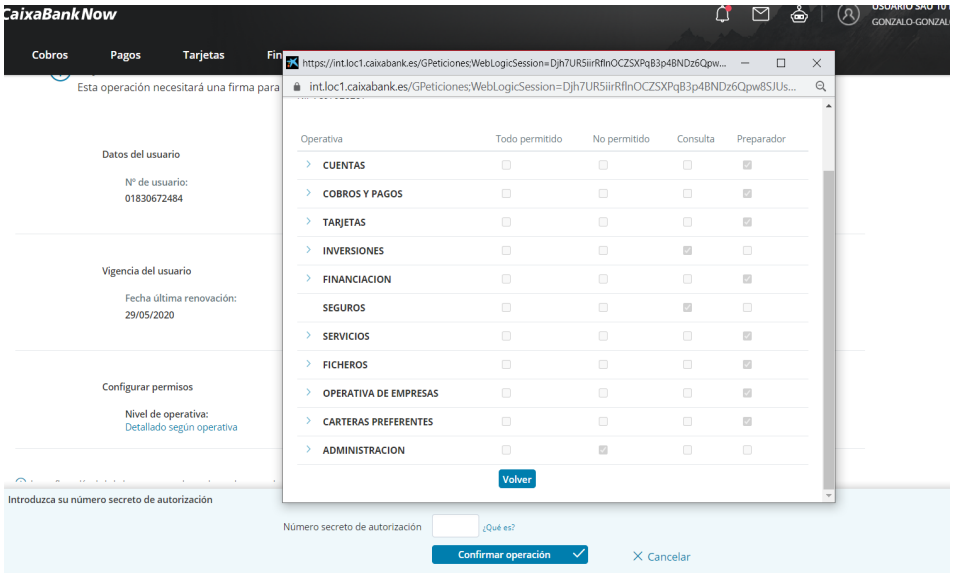


Figura 20: GUS - Ventana con permisos usuario detallados

Con el siguiente código generaremos un enlace, por el cual podremos usarlo para acceder al endpoint que hemos especificado. En la pantalla de administración de usuarios (figura 10) se recoge de back-end un array con los identificadores de cada uno de los roles que se han modificado, LLISTA_NODES_MODIFICADA. Por defecto, cuando se crea un usuario detallado y no se modifica ningún permiso de las diferentes operativas, se convierte en un preparador. Por eso se guardara solo los identificadores de aquellos que hemos cambiado ya que por defecto este tendrá el poder de un preparador.

```
beanMap.put(BeanFactory.newEnlaceBeanNormal("linkNodes",
                BeanFactory.newLiteralBeanDirecto(
                "/GPeticiones?PN=GUS&PE=4
                &CLICK_ORIG=NAV_GUS_21
                &LLISTA_NODES_MODIFICADA
                =" + llistaNodesModificada+
                "&DOCUMENT_IDENTIFICACIO=
                "+usuari.getNifUsuario()),
                BeanFactory.newLiteralBeanDirecto(""),
                BeanFactory.newLiteralBeanDirecto("")));
```

Listing 16: Código crear URL

Para realizar la llamada a dicho URL que generamos en el fichero java de la petición, haremos una búsqueda (por todo el documentos, incluido su raíz) de los elementos que tengan el mismo nombre de los elementos a buscar.

```
function listado(){
        var ta=document.getElementsByTagName('A');
        window.open(ta[ta.length-1],'lista',
                "toolbar=no,directories=no,
                status=no,scrollbars=yes,
                resizable=yes,menubar=no,
                width=550,height=550");
}
```

Listing 17: Código JSP funcion llamada pop-up

Inicialmente, se añadió una comprobación tal que existiera la variable ta (no nulo) y que este fuera mayor que 0 (su longitud), con el fin de que si se diera el caso de no encontrase ningún tagname que fuera un enlace, no abriría una ventana nueva. Vimos que esta comprobación era innecesaria debido a que en el fichero java ya hace una comprobación por si fuera el caso de que sea un usuario detallado y poder representar dicho rol en formato link (como se pude observar en la figura 19, ya que no es uno detallado).

En esta página se añadió un evento Tealium generado al acceder en la página. Al acceder, se generará las siguientes funciones (un script), las que hay en las figuras 21 y 22. Podemos observar que con la información que se le pasa por java en el fichero (PAGE_SECTION, PAGE_CATEGORY,PAGE-SUBCATEGORY1) generará el objeto digitalData autorrellenado con los parámetros necesarios.

```
function enviaEventApiTealium(datos){
if(typeof window.top.sendLink === 'function'){window.top.sendLink(datos);}
}

function loadTealium(){

var digitalData={
  page_section:'linea abierta',
  page_category:'gestion usuario',
  page_subcategory1:'listado',
  page_name:'linea abierta:gestion usuario:listado',
  page_language:'es',
  page_area:'privada',
  page_typology:'pagina',
  page_environment:'desarrollo',
  page_version:'normal',
  device:'desktop',
  client_id:'D2A444545E299DFEF4D767C859EFC70F',
  client_type:'cliente',
  client_login_status:'logado',
  client_segment:'particulares',
  heavy_digital:'no',
  ind_holabank:'false',
  ind_autonomo:'false',
  ind_negocios:'false',
  treatment_online:'true',
  consent_ccee:'true',
  consent_profiling:'true',
  form_maxloan_limit:'0'
};
```

Figura 21: GUS - Evento Tealium por la página, I

```
  if(typeof window.top.sendView === 'function'){window.top.sendView(digitalData);}
}

function TealiumAddLoadEvent(func){
        var oldonload = window.onload;
        if (typeof window.onload != 'function') {
            window.onload = func;
        }
        else {
            window.onload = function() {
                oldonload();
                func();
            };
        }
}

 TealiumAddLoadEvent(new Function('setTimeout("loadTealium();",50)'));
</script>
```

Figura 22: GUS - Evento Tealium por la página, II

El programa enviará el objeto digitalData en la función onLoad(), que esta función se ejecutará solo cuando toda la pantalla actual haya cargado todos el

contenido de ella. Esto quiere decir que será la última acción que se llevará a cabo de la pantalla.

11.3.2. Alta Usuario

Para comenzar en este flujo de la operativa, estuvimos mirando como estaba actualmente. A diferencia de los anteriores, en este había más pantallas por donde pasaría el usuario. Revisando todo el contenido, habían dos pantallas con las que se tenían que cambiar en respecto con el resto del recorrido de esta funcionalidad. Como no era del todo necesario cambiar todo, se decidió dejar para el final, en el caso de que hubiéramos acabado antes los objetivos de cada sprint. Además como las anteriores dos pantallas habían sido retocadas recientemente no había la necesidad de volver a retocarlas.

Primero de todo, añadimos un enlace volver en la parte superior, viendo la figura 24, motivado por el frame que esta situada en la parte de inferior de la pantalla. Esta parte, no esta directamente añadida en la página en si, sino que esta sobrepuesta por encima, es decir, se podría tratar como dos pantallas independientes. Este cambio es debido a que hay una idea importante sobre la experiencia del usuario (UX) que hay que tener en cuenta. *Los usuarios quieren conseguir alcanzar su objetivo con el mínimo esfuerzo posible.* Algo que puede resultar obvio pero que no resulta tan fácil como parece. Por ejemplo, en este caso si no estuviera el icono, tendríamos solamente la opción de cancelar que hay en el frame de clave de autenticación. Esto podría hacer pensar que esa opción sirva para cancelar la operación en la cual estamos ejecutando o podría hacer la función que volver a la página anterior. En el caso más cercano, baja de usuario (figura 17), podemos apreciar que no hay ningún enlace volver, debido a que tanto si quieres volver o quieres cancelar la operación, volvería a la misma pantalla. Este caso no se puede dar en esta situación a causa de que el flujo es más largo.

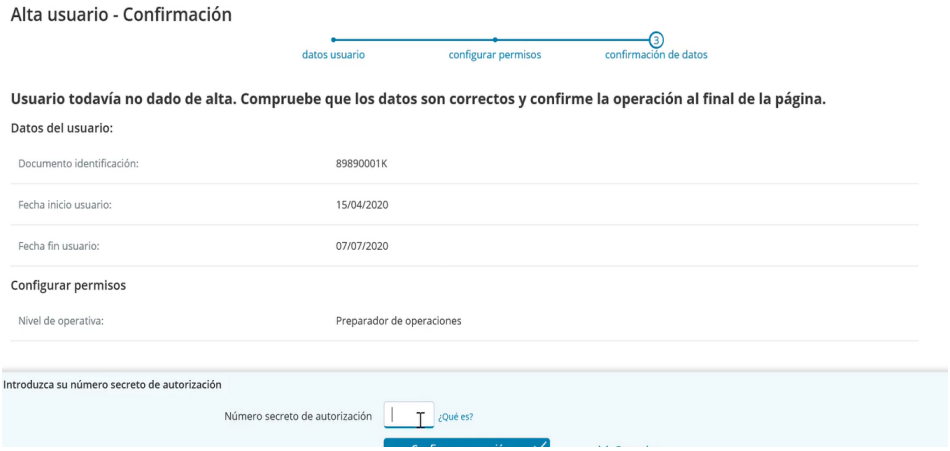


Figura 23: GUS -Confirmación alta usuario antiguo

Además se puede apreciar que, tanto como esta como en la operación anterior, hemos decidido añadir el icono de la impresora para poder imprimir un comprobante. A parte, decidimos dejar la barra de progreso que hay en la parte superior (la cabecera) puesto que es importante a que el usuario este en todo momento atento ya que sabrá donde está y por donde vá en el camino que ha de recorrer para realizar dicha operación. Siempre y cuando hayan distintos pasos para ejecutar la funcionalidad.

Por lo que respecta a las modificaciones, se decidió rehusar el formato anterior, el mismo template de baja de usuario, en la estructuración de los datos a mostrar del usuario. Por otra banda, al revisar lo que se muestra en la antigua vista de confirmación de alta del usuario, figura 23, se detecto que no mostraba mucha información, solo lo que se había rellenando en las pantallas anteriores. Se puede considerar que es lo justo y necesario, debido a que los pasos hasta este punto son primero de todo añadir el DNI del nuevo miembro donde se harán las comprobaciones pertinentes para saber si existe el usuario, las fechas correspondiente, tanto inicial como final, asignadas y el rol que tendrá usuario. Esto podría resultar confuso si se diera el caso de que el DNI fuera incorrecto, en el sentido de fuera otro miembro, y dieras permisos a uno que no quisieras. Por eso, usando la bolsa podemos añadir un nuevo parámetro que recorrerá durante todo el flujo en el que estemos. Por el cual no hará falta ir llamando al servidor durante todo este recorrido.

Es cierto que no disponemos del nombre del nuevo usuario en el que querremos dar de alta hasta el punto anterior, es por ello que añadiremos una nueva línea debajo de la llamada a la API.

```
lnkSON_GUS.executarConfigurarPermisos(caixaSession,
                                      logRegPres);
nomUsuari = lnkSON_GUS.getNomUsuari();
```

Listing 18: Código obtener nombre usuario

Al ejecutar executarConfigurarPermisos conseguimos crear un objeto usuario con sus características por el cual hemos estado añadiendo. Hay que decir que en este punto aún no se ha hecho una alta ni nada, sino que buscamos el usuario para encontrar la información personal que hay asociada a esa persona, tales como tipo de contrato (CLO) y el número de carpeta. A partir de esta situación, ya tendremos información necesaria para realizar una alta debido a que era necesario rellenar un mínimo de la información en realizar esa ejecución.

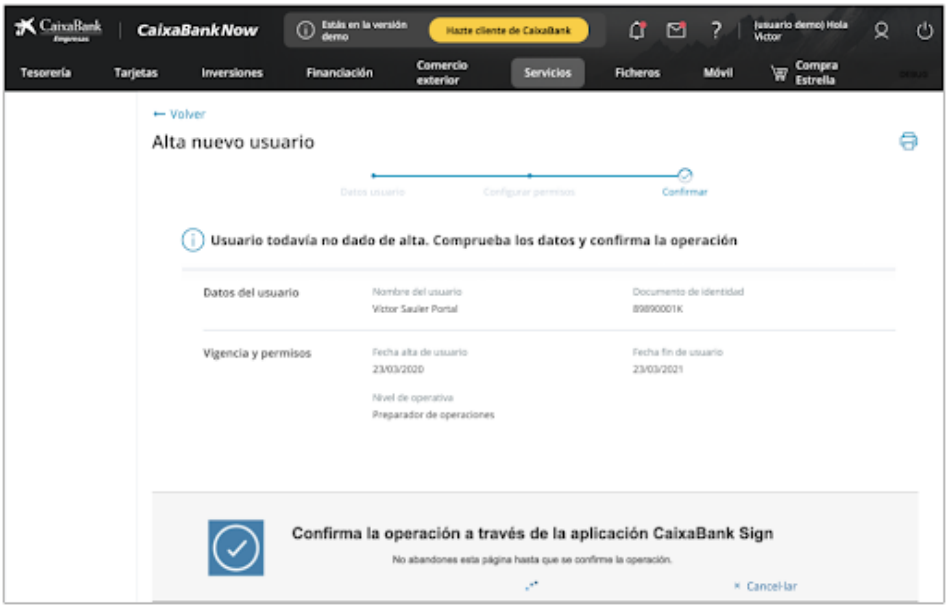


Figura 24: GUS - Confirmación alta usuario nuevo

Una complicación a la hora de desarrollar esta página (figura 24), no fue un problema severo sino algo que se tenía que tener en cuenta, fue que el desarrollador original que hizo esta página, reutilizó esta clase para que pasarán dos peticiones por esta. Me explico, anteriormente la página de consulta y de confirmación alta de usuario, se implementaba en la misma clase java pero en dos jsp diferentes. Entonces, habrán dos lógicas de negocio distintas en un mismo lugar.

Por otro lado, también se tomo la decisión de añadir el link en los usuario que sus permisos fueran detallados. Anteriormente, no se contemplaba la opción de muestra de la nueva ventana con esa información. La implementación fue igual que en el anterior caso solo que tuvimos que añadir el atributo donde se almacenaba estos permisos modificados, LLISTA_NODES_MODIFICADA, e ir traspasándola durante todo el recorrido de manera correcta.

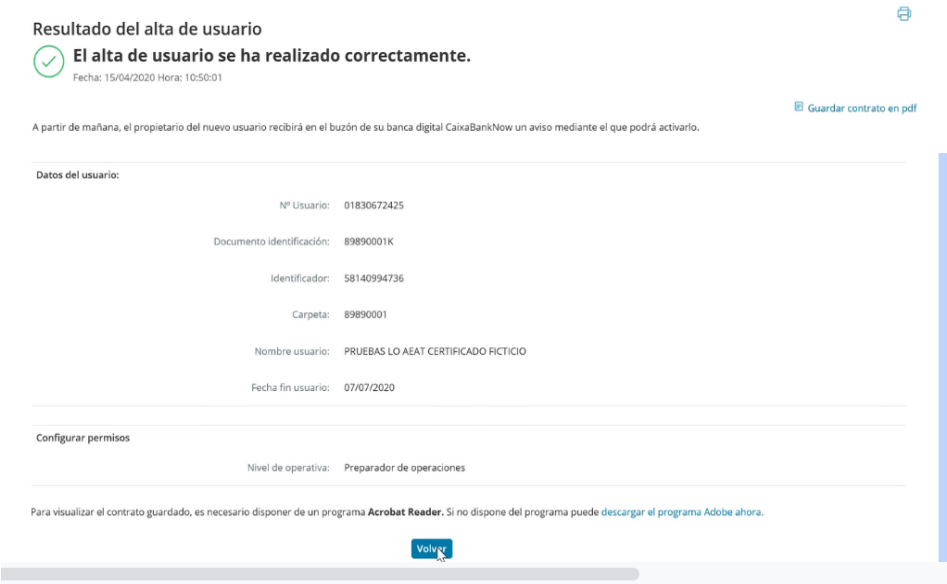


Figura 25: GUS - Resultado alta usuario antiguo

A la hora de formular el diseño de esta pantalla (figura 25), se uso el mismo que se ha llevado en las anteriores, únicamente en la estructuración de los datos del la persona. No tendría mucho sentido ir cambiando la manera de estructurarlos, por el motivo de que para el usuario le resultaría un poco lioso tener que ir entendiendo la estructuración por cada una de las páginas. Por eso siempre se intenta a la medida de lo posible, organizarlo de la misma manera.

Aquí se puede observar, viendo la figura 25, el problema de los taglibs comentado al inicio. Se puede ver como el enlace de guardar contrato en pdf esta por encima de la linea, y no a la misma altura que la frase que empieza por a partir de mañana. Eso es debido a que toda esa zona, se puede ver como si fuera el tag que hemos creado y no se pudiera añadir cualquier otra cosa, ya que se trata como si estuviera ocupado por un componente. En la reestructuración se decidió ponerlo debajo de la cabecera y tenerlo más centrado a la vista del usuario, motivado por reutilizar ese componente.

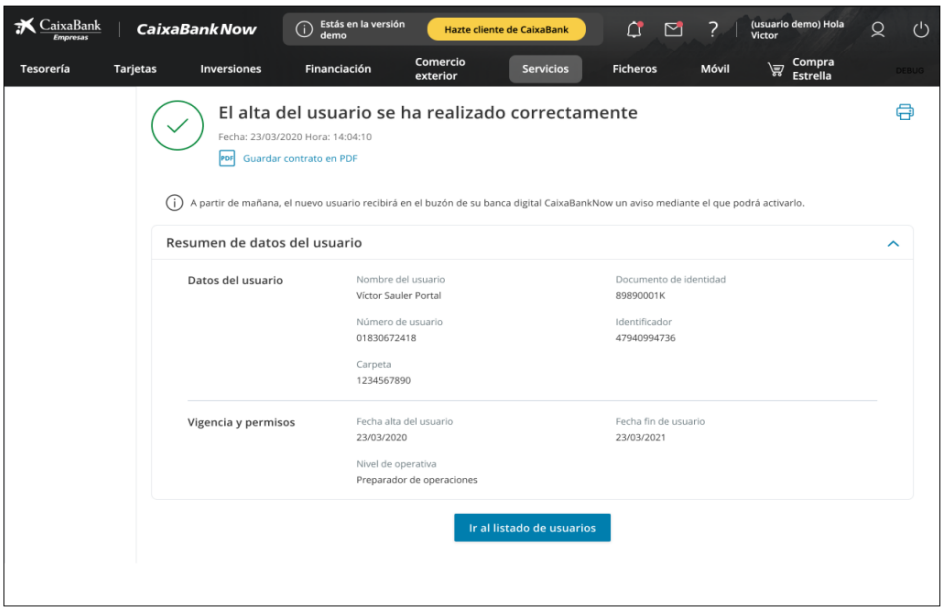


Figura 26: GUS - Resultado alta usuario antiguo

Al ser esta la página final de alta de usuario de todo el proceso, fue mejor visto que el usuario visualizará toda la información relacionada a este, en lo que respecta en el tema de administrativo.

Por otro lado hay que comentar que si al dar de alta un usuario con fecha de caducidad sin definir se pondrá en su lugar un literal donde informará lo siguiente, tanto en la pantalla de confirmación como resultado de la alta:

Figura 27: GUS - Fecha sin fin detallado

11.3.3. Consulta Usuario

Cuando se empezó a analizar dicha funcionalidad, se comprobó que no había diferenciación a la hora de distinguir las dos principales relaciones que existían,

apoderados o autorizados. Debido a que con anterioridad no se contemplaba esta separación que hemos aplicado de un principio aprovecharemos en adaptarlo ahora, ya que si eres uno u otro hay una diferenciación sobre las funciones a realizar.

En las siguientes figuras, las 28 y 29, podemos ver las vistas las pantallas actuales de las operación de consulta. La única diferencia entre las dos pantallas solamente residen en si está caducado o es vigente y no se se caducará en las próximos 30 días.

Asimismo, decidimos reutilizar los beanMaps de las clases java a la hora de encapsular la información para que la jsp los pudiera recibir y tratarlos como se deberían. Es por ello que para todos estos casos de la consulta, se hizo toda en una misma jsp. Todas esta lógica que se implementará estará desarrollada en la misma clase java y se tratarán las diferentes posibilidades.

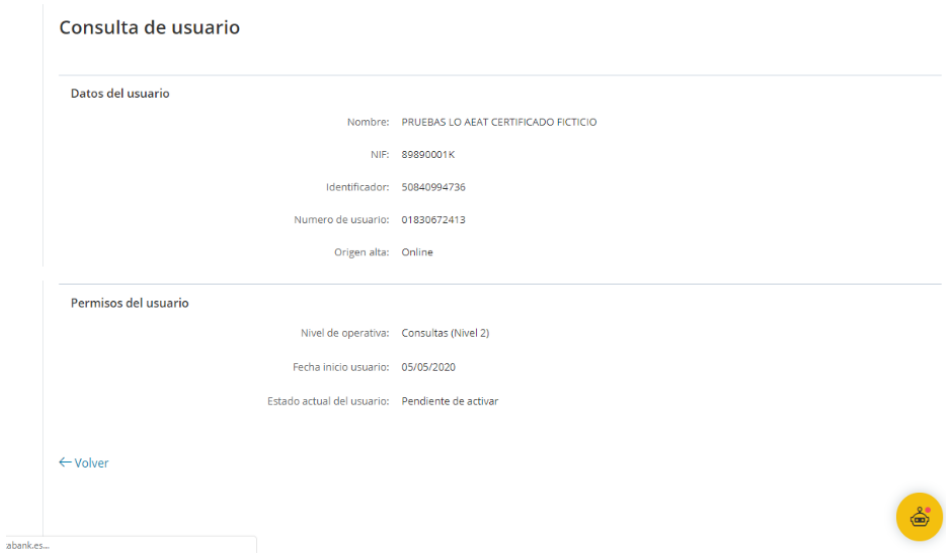


Figura 28: GUS - Consulta usuario antiguo

Figura 29: GUS - Consulta usuario a punto de caducar antiguo

La diferencia de un usuario que esta a punto de caducar y de uno que ya esta caducado solamente reside en un simple detalle. En el atributo *Estado actual del usuario* pasaría a tener como valor el string *Caducado*.

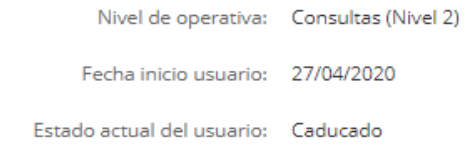


Figura 30: GUS - Consulta usuario caducado antiguo

Como resultado, se decidió realizar varias vistas dependiendo del rol que tenga. Aparte de hacer una remodelación de la pantalla se añadió nuevas acciones a ejecutar (no en todas), que comentaremos en los apartados siguiente.

Usuario Apoderado

Primero de todo comentaré sobre la consulta de un usuario apoderado. Como he ido comentando durante todas las mejoras realizadas de las plantillas, seguiré utilizando la misma manera de estructurar los datos relacionados sobre los usuarios.

En un principio, en el SOT ibamos a contemplar la recogida (en la clase OUT) de un nuevo parámetro que es la empresa a la cualquier pertenece, por eso debíamos modificarla y tener en mente de que back-end devolvería un nuevo parámetro para nuestro uso. Pero al final no se hizo así, ya que al analizar más en profundidad deducimos que los usuarios al iniciar una sesión en la aplicación se guardaría esa información, por el cual no sería necesario almacenar dicha información repetida.

```
LinkLoe linkloe = (LinkLoe)caixaSession.getElementoBolsa
                                    (LinkLoe.NOMBRECLASE);

hmDatosSalida.put(ConsGUS.NOMBRE_EMPRESA,
                                        linkloe.getLinkNombreEmp());
```

Listing 19: Código obtener información de la sesion actual

Por otra banda, siguiendo los pasos anteriores añadimos la opción de añadir la función de imprimir la pantalla de la consulta al usuario que queremos, algo importante que casi no tenía presencia en las pantallas.

Los enlaces volver, los pasaremos a la parte de superior debido a que al ser más larga la estructuración de los datos del usuario, no le obligaremos a tener que recorrerlo, aunque sea mínimo. Un ejemplo práctico sería el caso de que una persona se equivocase al consultar un usuario. Este enlace estaría a primera vista y podría volver a la pantalla inicial para realizar la consulta de nuevo sobre el que tenía pensado ejecutar.

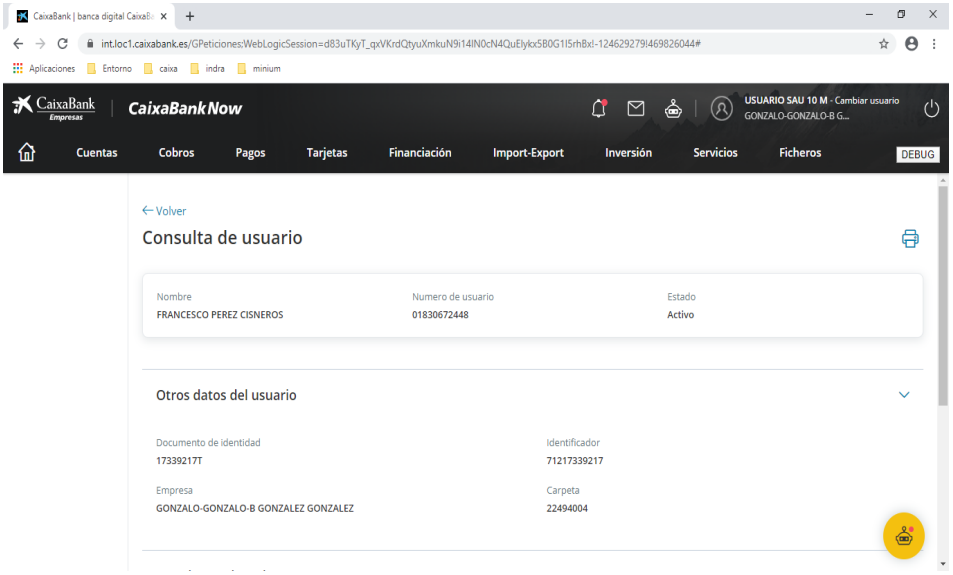


Figura 31: GUS - Consulta usuario apoderado nuevo I

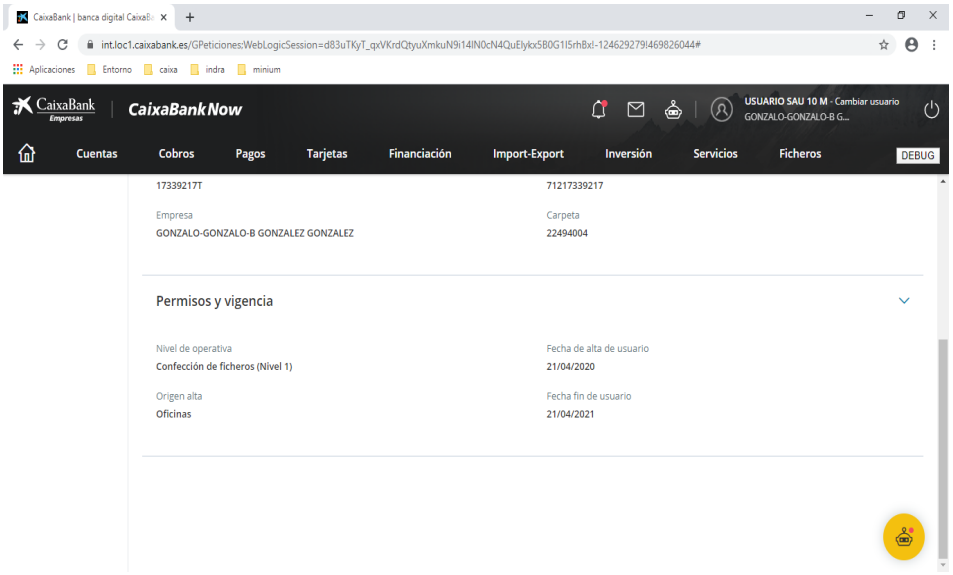


Figura 32: GUS - Consulta usuario apoderado nuevo II

Por lo que respecta en los otros aspectos, vamos a seguir el mismo patrón que hemos utilizado hasta ahora, pero no van a ser taglibs sino que incrustaremos

código HTML para poder reestructurarlo a nuestra manera. Por ello utilizaremos una clase StringBuffer que nos permite modificar y aumentar el tamaño de este. Lo normal sería usar un String, pero como el valor del objeto va a ir cambiando un gran número de veces y solo será modificado por un mismo hilo o thread, utilizaremos la clase comentada. Por otra parte existe otra clase idéntica a esta, pero que es más útil en los casos que exista la necesidad de la sincronización (thread safe), si el objeto es modificado por varios threads. Pero como no es el caso, usaremos el StringBuffer.

```
private static final String ABRE_DIV = "<div ";
private static final String CIERRA_DIV = "</div>";
....
StringBuilder textTemp = new StringBuilder();
textTemp.append(ABRE_DIV + ABRE_CLASS + C_BOX
                + GUION_MEDIO_DOBLE + "block"
                + CIERRA_COMILLA + CIERRA_ETIQUETA);

textTemp.append(ABRE_DIV + ABRE_CLASS + PADDING
            +GUION_MEDIO + NORMAL
            + CIERRA_COMILLA + CIERRA_ETIQUETA);

textTemp.append(ABRE_DIV + ABRE_CLASS + U_FLEX + SPACE
            + U_FLEX + GUION_MEDIO + LLAUTO
            + SPACE + U_ROW + GUION_MEDIO_DOBLE
            + SEP + GUION_MEDIO + VALOR_10
            + CIERRA_COMILLA + CIERRA_ETIQUETA);

textTemp.append(ABRE_DIV + ABRE_CLASS + U_FLEX
            + GUION_MEDIO + LLAUTO + SPACE
            + U_FLEX + SPACE + U_ROW
            + GUION_MEDIO_DOBLE + SEP
            + GUION_MEDIO + VALOR_10 + SPACE
            + U_FLEX + GUION_MEDIO + WRAP
            + CIERRA_COMILLA + CIERRA_ETIQUETA);
```

Listing 20: Código Java StringBuilder

Esto es una parte del la construcción de la cabecera de los datos del usuario donde todos los parámetros que se le asocia a la clase builder estarán ya asignadas . En la parte de abajo se puede ver cómo se representaría el código de arriba en el el fichero jsp.

```
<div class="c-box--block">
<div class="padding-normal">
<div class="u-flex u-flex-11auto u-row--sep-10">
```

```
<div class="u-flex-11auto u-flex
    u-row--sep-10 u-flex-wrap">
```

Listing 21: Código del SringBuilder en JSP

Usuario Autorizado

En el caso de que un usuario fuera autorizado, había un cambio relevante respecto al anterior. Como se había comentado en los nuevos casos de uso, un usuario apoderado podía realizar cualquier tipo de operación, pero con la única condición de que afectaba a los usuarios autorizados, ya que entre apoderados no se pueden realizar ninguna operación. Es por ello, que añadimos un cuadro en la parte de la derecha con un acceso directo a las diferentes operaciones.

Con este cambio conseguimos que el usuario pueda realizar las operaciones una vez haya comprobado toda la información del usuario y no tenga que hacer varios clics de más para realizar la operación. Algo que ayudaría a agilizar las operaciones sobre los usuarios. Siempre hace falta comprobar antes de realizar cualquier tipo de operación, mucho más si se trata de administrar usuarios.

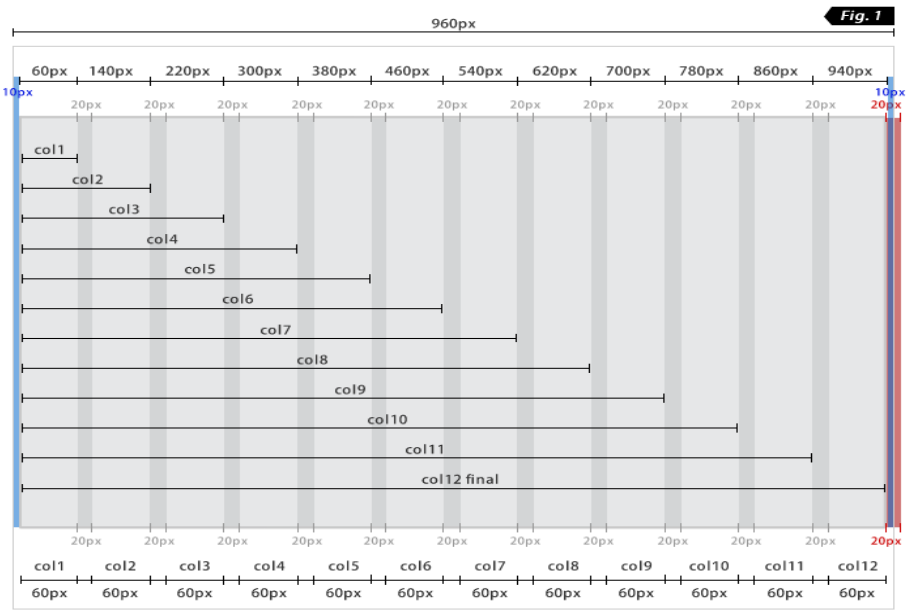


Figura 33: GUS - Framework CSS

Viendo la representación de la página según el framework de CSS, podemos ver como se tiene en cuenta la subdivisión de una pantalla. Por ello, con la modificando los tags de los divs de la view conseguimos poder dividir cada agrupación. Para ello, dividiremos en dos secciones la zona correspondiente a la nueva parte de este tipo de usuarios. En concreto :

- el primer div: u-col–width-9/12c, donde corresponde a las primeras 9 columnas de la figura 33.
- el segundo div: u-col–width-3/12c, donde representa las últimas 3 columnas de la figura 33.

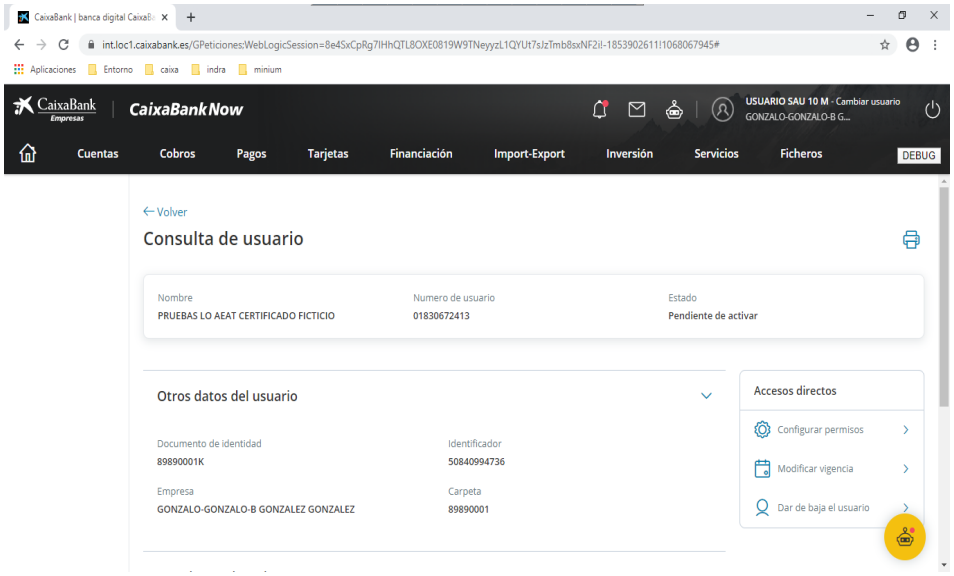


Figura 34: GUS - Consulta usuario autorizado nuevo I

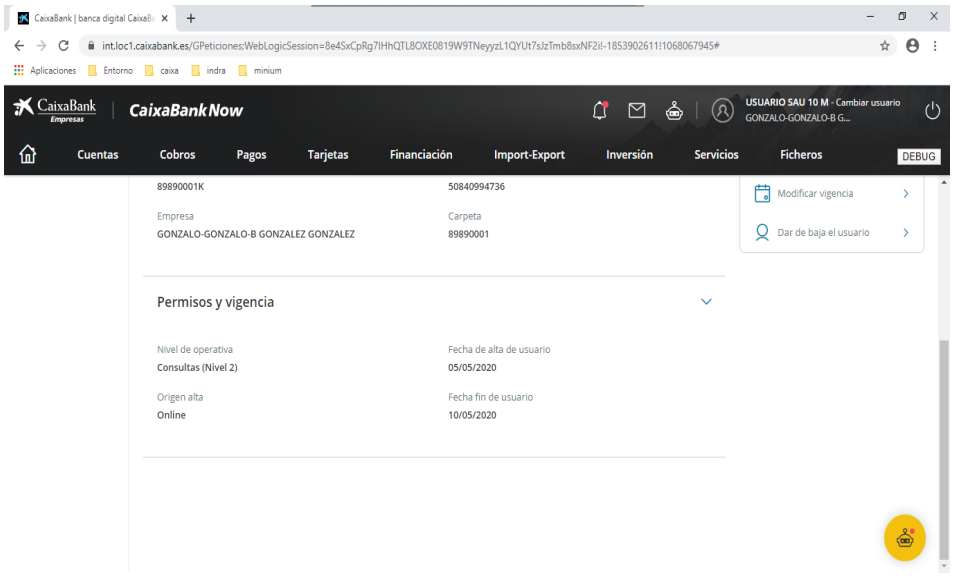


Figura 35: GUS - Consulta usuario autorizado nuevo II

Usuario Apoderado Caducado

Tanto en el caso de los usuarios apoderados como en el de los autorizados, si se diera el caso en el que esa persona esté caducado o apunto de caducar, añadiremos un enlace por el cual podremos realizar la petición de la renovación de este.

Por consiguiente lo único que deberemos hacer es comprobar el estado de usuario y si se diera el caso de la necesidad de poder renovarlo, aparecería el botón con el redireccionamiento a la petición de renovar.

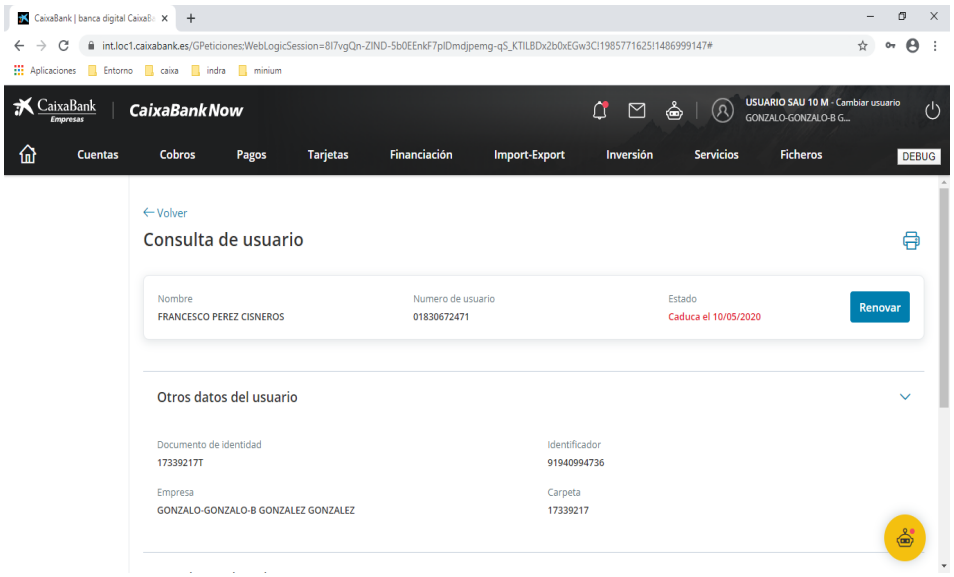


Figura 36: GUS - Consulta usuario apoderado caducado nuevo

11.4. Redireccionamiento de enlaces volver

En esta parte del proyecto empezamos a realizar las modificaciones en todo lo relacionado con la lógica de negocio, y ya dejando de la lado lo estético.

Cabe recordar que hemos creado una nueva pantalla para mostrar la nueva manera de listar los usuarios. Por ello deberemos que modificar las funciones de volver que habían anteriormente, escoger a qué petición (PE) va a volver. Igualmente, hay que mantener las redirecciones a las pantallas antiguas debido a que si falla el flujo principal, este deberá seguir el antiguo.

Antiguamente, como solamente había una pantalla donde se llamaban a las operaciones que tratamos, la función estaba hecha de manera que estaba hard-codeado en los destinos de petición. En otras palabras, se ponía directamente el valor de la petición a la que tenía que ir.

Actualmente, se hará de una manera que si se crean nuevas pantallas de por medio o hay otras operativas que lo llaman, no habrá necesidad de tener que cambiar las funciones de las pantallas donde haya el enlace. Esto resolverá problemas más adelante al no tener que ir toqueteando todas esas pantallas que fueran afectadas por las futuras decisiones.

```
beanMap.put(BeanFactory.newInputBeanPredefinido
                ("InputTORNA_PN", ConsGUS.TORNA_PN, "GUS"));

beanMap.put(BeanFactory.newInputBeanPredefinido
                ("InputTORNA_PE", ConsGUS.TORNA_PE, "27"));
```

Listing 22: Código añadir información a Bolsa

Para empezar se tendrá que enviar un parámetro desde las primeras pantallas, un lugar claro son aquella donde se hacen las peticiones a las operaciones que estamos trabajando. Durante todo el recorrido se irá tratando ese parámetro y enviarlo por el flujo en el que estamos.

Dentro del contexto de la página JSP tenemos disponible la variable *request*. En ella, tendremos un Hashmap, llamada bolsa, que creamos en la clase java que le pasaremos y consecuentemente tendremos disponibles los parámetros y valores que nos interesan.

```
bolsa = (HashMap)request.getAttribute("Bolsa");
String PN = (String)bolsa.get(ConsGUS.TORNA_PN);
String PE = (String)bolsa.get("TORNA_PE");

...

f.PN.value=<% PN %>;
f.PE.value=<% PE %>;
```

Listing 23: Código en JSP pasar información Bolsa

En este caso, tenemos las tres primeras líneas del código con el cual recogemos lo que se ha guardado durante el proceso y dentro de ese componente tendremos disponibles los valores de los parámetros nuevos. Estos valores nos indicará dónde enviar la próxima petición, por el cual los añadiremos en la variable de javascript donde ejecutará la función submit() ofrecida por un objeto HTML.

11.5. Modifciación lógica negocio

En las subsecciones de esta sección, operaciones, hemos hablado bastante sobre el funcionamiento y la remodelación que hicimos sobre las pantallas de cada uno. Es cierto que en el apartado de casos de usos explico como es actualmente, sin embargo en este explicaré lo que hay por detrás de todo los cambios.

Es cierto que, al principio del desarrollo se decidió que fuera front-end el que tuviera el control de la nueva lógica y que hiciéramos las comprobaciones pertinentes en todo momento. Pero al cabo de un tiempo, cuando ya teníamos acabado y entregado toda esta parte del trabajo nos comunicaron que al final sería back-end el que tendría el control.

Al ocurrir esto, se tuvo que tocar los SOT, que al fin y al cabo es el lugar donde se trata las transacciones y los datos de recogida de una petición a la parte del servidor.

Para ello, primero de todo era modificar la clase contenedera de los parámetros, con los nuevos atributos que nos devolvia back-end pero que anteriormente no se contemplaban, por consecuencia no se guardaban ni se trataban.

```
public class ContratosLA {
        private Number numUsuarioContrato;
        private String dniApo;
        private String persona;
        private Number numPerUsuario;
        private String tipoApoderado;
        private String nivelUsuario;
        private String nivelOperativo;
        private Number codigoRelacion;
        private String relacion;
        private Number ordenRelacion;
        private String fechaRenovacionAutorizados;
        private String fechaRenovacionAMostrar;
        private String fechaCaducidadUsuariosOnLine;
        private String fecha;
        private String estadoUsuario;
        private String estado;
        private String codigoIndicadorVisibilidad;
        private String IndicadorVisibilidad;
```

Listing 24: Clase ContratosLA

Despues de las modificaciones, esta fue la clase definitiva de contratoLA donde contiene todos los datos, más las funciones getters y setters, que nos devuleve la prestación de la llamada a back-end. Hay mucha más información ya que de esta manera conseguimos reducir otras llamadas a la parte del servidor que se realizaban al pedir información sobre un usuario. Por eso se generará una objetivo tipo Lista, que estará en la clase Out, donde en su interior tendrá estos usuarios.

```
public class Out_ListaUsuarios {
        private boolean hayError;
        private String mensajeError;

        private String dniTit;
        private String nomTit;
        private Number numTit;
        private String tipoTit;
        private String estadoContrato;
        private Number claveContinuacion;
        private boolean hayContinuacion;
        private TipoLista listaContratosLA =
                                new TipoLista();
```

Listing 25: Clase Out_ListaUsuarios

Este es la clase que se retornará a nuestro PN un vez se haya ejecutado la llamada y se haya finalizado de manera correcta. De manera que si la devolución de back-end no es nulo significará que ha ocurrido un problema en la llamada.

```
strEstado = Mensaje.enviarAbsis(logRegPres,
            mensajeInAbsis, mensajeOutAbsis,
            servicioSolicitado, caixaSession, demo);

if (strEstado == null) {
            this.setLnkOut_ListaUsuarios
            (mensajeOutAbsis.getLnkOut());
```

Listing 26: Código comprobación y envio de los datos

En el siguiente fragmento podemos ver como al ejecutar la función del SON executarLlistaUsuarisSINuevo(), hará sus repectivas operaciones para recoger el listado que hemos explicado más arriba, la clase ContratosLA, y a partir de alli podremos trabajar con los usuarios. Dentro del bucle for() recorreremos uno a uno cada usuario e iremos haciendo todo aquello para poder mostrar la figura 13 (el listadi de usuarios de la administracioón de usuario) y gestionar toda la información relativa a las futuras operaciones.

```
lnkSON_GUS.executarLlistaUsuarisSINuevo
                (caixaSession, logRegPres);
listaContratosLA = lnkSON_GUS.getLnkLlistaUsuaris()
                .getListaContratosLA();

for(int i=0; i < listaContratosLA.size(); i++){
```

```
...
}
```

Listing 27: Llamada y recogida de la petición al SON

Una vez en nuestro PN y ya habiendo gestionado los datos, podemos ver que la clase contratoLA tenemos un string IndicadorVisibilidad que con ello dependiendo del valor que tengan almacenado, se podrá finalizar la ejecución o no. Se añadirá una comprobación con esta condición donde si no es igual conllevará que no se podrá finalizar dicah operación.

```
(entrada.get("VISIBILIDAD")).equals("M")
```

Listing 28: Condición de saber si puede finalizar una operación

El resultado de añadir esta condición y en el momento que nos devolviera un NOK, mostraria una pantalla con el error pertinente como se puede ver en la siguiente figura:

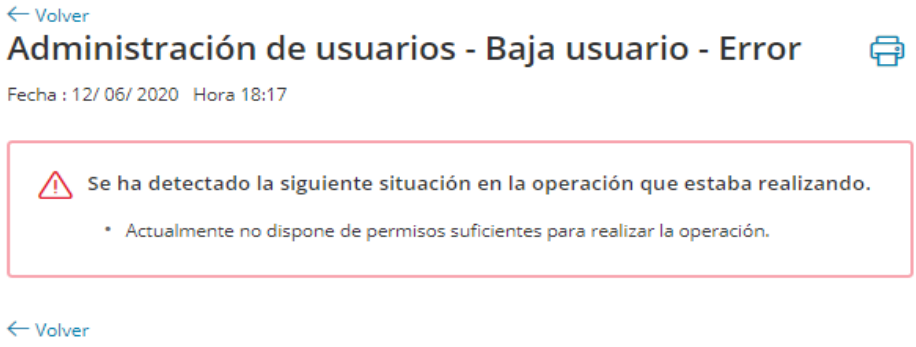


Figura 37: GUS - Operación denegado por permisos

Esta comprobación que se realiza antes de finalizar la operación es distinta que la que he comentado en la sección de casos de usos. No tiene nada que ver el motivo de fallida de dicha funcionalidad. Es cierto que tiene que pasar estas dos validaciones, pero uno es completamente distinta que la otra.

12. Problemas durante el desarrollo

Durante el desarrollo de las mejoras se tuvo varios problemas, ya fuera por la parte de front-end (nuestra parte) como por parte de back-end, que allí ya deberíamos comunicarnos con el equipo correspondiente. Por el cual a veces nos contestaban de inmediato y otras veces debíamos dejarlo apartado hasta su respuesta.

12.1. Entendimiento de la Bolsa y el Flujo

Durante el inicio del proyecto, fue bastante difícil entender el comportamiento de la bolsa. De manera teórico era entendible, era un contenedor que se iba pasando a través de las peticiones por el flujo de negocio. A la hora de implementarlo fue más caótico porque se tenía que tratar siempre y si en una parte no lo tenías en cuenta no pasaba de esa petición. Habían bastantes clases por donde recorrían estos parámetros y se tenía que entender al pie de la letra.

12.2. Usuarios detallados con back-end

Durante el proceso pudimos descubrir que varias altas de usuarios, en concreto si fuese el rol detallado. El problema se daba cuando se quería ejecutar una operación sobre él. Por lo que se vio, el resultado al hacer la alta no retornaba error, pero cuando se accedía a la base de datos para buscarlos nos devolvía error el SON que lo ejecutaba. En este caso encontraron dos tipos de problemas:

- El primero por parte de back-end no guardaban bien la lista de los roles modificados, allí no pudimos hacer nada más que comunicarles el mapeo que hacíamos para almacenarlos y traspasarlos.
- El segundo problema era cuando no se daba una fecha final a la hora de realizar el alta. Anteriormente, se enviaba un valor nulo y cuando se pedía el resultado no se tenía en cuenta que se podía almacenar un valor nulo.

12.3. Usuarios detallados link privilegios

Trato este problema separado respecto al apartado anterior debido a que no era del mismo estilo. Mientras que en la otra subsección se habla sobre la comunicación entre front y back, aquí se comentará solo un problema de front-end.

En el desarrollo de la ventana modal con listado con los diferentes permisos de los usuarios detallados, usamos el objeto Window. Por ello, en la función open() de dicha instancia se le añade la URL destino a donde queremos ir. Debido a esto, primeramente se hacía un búsqueda de los tags de link que fuesen enlaces.

En el código java creamos un enlace que lo pasamos por un beanMap para que la petición en cuestión pueda ser consultada en su respecto clase jsp. El problema venía cuando habían más tipos de dirección, ya que las funciones javascript en dicho fichero los representaba como si fuesen un tag de enlace. Por el cual, no lanzaba nada motivado a que no había una URL en dichas funciones.

12.4. Firma de autentificación

Como hemos comentado anteriormente, el recuadro donde aparece el código de autenticación, que se considera como la firma del usuario, en las pantallas donde se ejecuta la operación en back-end se considera un frame a parte de la pantalla principal. El problema encontrado era al realizar el volver que se encuentra en la pantalla original que seleccionaba como target el frame del recuadro de la autenticación.

Para ello se encontró un atributo de HTML donde se podía especificar el destinatario. Encontramos que el problema era debido al frame solapada sobre la pantalla. Por defecto, el valor del atributo que se asigna es _self, que despliega el resultado en el frame actual, en este caso el recuadro. Por ello, se modificó esa variable para que fuera a la pantalla "padre", asignándole el valor _parent.

Se creó la siguiente función javascript enlazado al botón de volver que resolvería este problema. Con la opción parent provoca que el documento sea mostrado en el frameset padre del frame actual.

```
function tornar(){
    var f=document.forms["PIE_II"];
        f.PN.value = 'GUS';
        f.PE.value = '3';
        f.target = "_parent";
        f.CLICK_ORIG.value='VOL_GUS_5';
        f.submit();
}
```

Listing 29: Código modificar target

12.5. Taglibs

Uno de los problemas que he ido comentando en el transcurso de este trabajo, han sido los taglibs. Es algo que facilita a los desarrolladores el uso de esto, debido a que en el java se podría crear una clase que centralizarse toda las creaciones de los contenidos, ya sean para la creación de tablas, los enlaces, etc.

El problema es que todo el espacio que ocupa ese componente, hago referencia a los espacios que se dejan en lo ancho de este, no se podrán reutilizar debido a que todo ese espacio, tanto como lo que se utiliza como lo que no está en uso, lo trata como si fuera del objeto.

Durante el proceso de resolución del problema, se intentó añadir dos estructuras en un mismo taglib, pero no había manera de que pusieran en paralelo una del otra. Al añadirlo se colocaba el nuevo componente justo debajo del primero.

12.6. Falta de comunicación por parte del cliente

Uno de los problemas más comunes durante el proyecto era la comunicación con el cliente. Es cierto que no se trata de un problema que debamos resolver buscando información o que debamos pensar cómo podríamos hacerlo con más eficaz. cada dos semanas se hablaba con ellos y estaban al día de todas las modificaciones o mejoras que se estaban realizando, pero por su parte no siempre había la comunicación necesaria. Un ejemplo de esto fue sobre quien el control de saber quien podía finalizar las operaciones. Principalmente, fueron asignadas a nuestro equipo, pero semanas más tarde decidieron que fuera a través de back-end quien lo tratara.

Esto provocó que en otro sprint tuviéramos que realizar los cambios pertinentes y modificar la comunicación de las mensajerías con la parte del servidor.

Otro caso a tener en cuenta fue en la primera pantalla de la administración de usuario. Durante el sprint planning se estuvo comentando sobre el menú desplegable de las operaciones a realizar. Primeramente querían esconder las opciones en el menú desplegable de aquellos que no tenían los permisos suficientes para finalizar la operación. Siguiendo el manual de la usabilidad, no se deben esconder las propiedades que se tienen, incluso si no se tuviera permisos. Más adelante, decidieron hacer caso al manual y nos pidieron que lo volviéramos a cambiar.

Estos dos casos son un claro ejemplo de los problemas que esmento en esta subsección. No es algo que nos perjudica de manera muy grave en el desarrollo, pero es algo que nos quita tiempo en modificar algo que ya se había hecho.

13. Desviaciones

A causa del desconocimiento de las tecnologías que se utilizaron para realizar este proyecto, al principio tuvimos un problema a la hora de implementar las tareas. En los primeros sprints tuve que mover las tareas para las siguientes debido a la inexperiencia y el poco conocimiento que tenía. Por este motivo se tuvo que modificar en los apartados de Análisis y Desarrollo e implementación de la tabla 1 que habíamos propuesto de un inicio.

Código	Nombre	Tiempo	Dependencia
T2	Desarrollo e implementación	370 h	T1
T2.1	Desarrollo de las funcionalidades	150 h	T1.3
T2.2	Implementación de componentes	150 h	T2.1
T2.3	Nuevos servicios	30 h	T2.2
T2.4	Testing	40 h	T2.3

Tabla 8: Resumen Tareas con las desviaciones

Por ello tuvimos que añadir 40 horas extras para el desarrollo e implementación. En la tabla 8 podemos ver una representación de la parte de la tarea en cuestión. Todo y que se necesitan 40 horas, esto no afectaría al coste del proyecto debido que añadimos un porcentaje de más al estimar nuestro presupuesto inicial (8.97 €/hora * 40 = 356.4 €).

En lo que respecta al Gantt propuesto inicialmente, figuras 2,3,4 y 5, hubo una modificación en lo que respecta a las tareas a realizar en ellas. Inicialmente se decidió estructurar de otra manera, por el cual primero se implementan las bases de la página y luego ir desarrollando los otros componentes. Pero al final decidimos hacer toda la pantalla, incluyendo todo lo contienen, junto a la implementación de la página. Por el cual, tendríamos toda la pantalla nueva acabada en cada sprint.

14. Posibles Mejoras Tecnológicas

Comentaré algunos aspectos que creo que mejorarían la aplicación tanto tecnológicamente como la manera de trabajar mejorando aspectos determinados.

14.1. React Native

Una de las posibilidades sería usar el framework de React Native, en vez del actual que es el Sprint, basado en las librerías de ReactJS y JavaScript.

La principal ventaja de ReactJS es poder generar el DOM ("Modelo de Objetos del Documento", que es la estructura de los elementos que se generan en el navegador web al cargar una página) de forma dinámica. Esto permite que para poder visualizar los cambios de los datos, no es necesario renderizar toda la página de nuevo, sino solamente el componente que haya sido actualizado. Cuando React compara el DOM Virtual con el DOM del navegador sabe perfectamente qué partes de la página debe actualizar y se ahorra la necesidad de actualizar la vista entera. Gracias a esta característica mejora: la experiencia de usuario al navegar por la aplicación web, la rapidez en la carga de las páginas y facilita el mantenimiento de la aplicación.

Por otro lado, existe una librería de React Native que nos permite tener nuestros componentes de React Native funcionando en la web, ya que inicialmente está pensado para el uso de Android y IOS (aplicación móvil). Hay que comentar que no solo la propia compañía es la que está trabajando en esto, sino que también existen terceros dónde están generando módulos para poder ejecutar módulos las aplicaciones en la web. Esto nos llevará una gran ventaja debido que actualmente hay distintos equipos para desarrollo para plataforma web como para dispositivos móviles. Con esta tecnología supondría un ahorro ya que el propio React renderiza los componentes dependiendo del dispositivo que estuviéramos mirando.

14.2. Github

Por lo que respecta a los controles de versiones, actualmente se utiliza SVN(Subversion), se podría utilizar el Github. Es bastante recomendable ya que es un sistema distribuido, lo que significa que, aunque existe un repositorio central en el cual se incorporan los cambios, todos los usuarios pueden descargar su propia copia de trabajo. De esta forma, todos tienen acceso al repositorio completo, incluyendo el historial local, sin depender de ningún tipo de conexión de red.

Por otro lado, Git identifica las acciones mediante una suma de comprobación (checksum) que impide cambiar los contenidos de cualquier archivo o directorio sin que se entere. No puedes perder información durante su transmisión o sufrir corrupción de archivos sin que Git lo detecte. Este contiene un historial de cambios completos, tanto locales como de las del repositorio.

14.3. Google Tag Manager

Las dos tecnologías, en nuestro caso utilizamos Tealium, ofrecen una funcionalidad similar en términos de configurar las etiquetas y registrar los datos. Pero una de las principales características es la increíble opciones en las variables y opciones de los triggers.

Estos dos componentes son donde de verdad brilla realmente GTM. Aproximadamente hay 10 tipos de triggers diferentes y casi 20 tipos de variables, por el cual hay una amplia gama de opciones a escoger.

Está constantemente actualizando y mejorando las herramientas. Al ser de Google, siempre presentan nuevas funciones, tipos de variables, de triggers y templates actualizadas.

Por otro lado, este es totalmente gratuito, solo hace falta tener cuenta de Google para poder utilizarlo.

15. Conclusiones y Futuro Trabajo

15.1. Futuro Trabajo

Como trabajo futuro hay algunos aspectos que hay que modificar en la parte de front-end. Sobre todo en las otras operaciones que hay en la operativa en cuestión. Aparte habría que indagar más sobre las implementaciones que hay detrás de cada una de ellas.

Por otra parte, habría que actualizar los servicios (SO) debido que en algunos casos no se utilizan todos los parámetros de entrada o que no se debería tener en cuenta debido que al iniciar sesión con un usuario ya tendremos algún tipo de información. Además, intentar conseguir un mayor ahorro en las llamadas a realizar al servidor, unificarlos de una manera. Conseguir que con una llamada, se pueda tener la información en vez de tener que ir llamándola varias vez. Esto conlleva modificar los métodos de la API ya desarrolladas.

Además se tendría que automatizar las pruebas en vez de hacer las pruebas de manera manual, como es el caso actual.

15.2. Conclusiones

La oportunidad de trabajar en este proyecto me ha ayudado bastante a la hora de introducirme a lo que es estar trabajando en equipos grandes, no solo con los que estoy diariamente sino también con los otros que trabajan en otras operativas. La comunicación es muy importante y este proyecto me ha ayudado ha fomentarlo.

El objeto principal era mejorar la aplicación web, tanto la parte funcional como los estilos, mediante un framework propio, parecido al spring, utilizando tanto Java como javascript como principales lenguajes de programación y la API de mensajería creado por la empresa para acceder a la parte de los datos. Después de finalizar el periodo asignado, podemos decir que el trabajo se realizó exitosamente. Aunque no todas las modificaciones que realizamos están ya actualizadas, debido que deben pasar un proceso antes de que estén hechos los cambios.

También, me he podido introducir en el mundo de la aplicación web y lo que respecta estar en la parte de front-end y la comunicación con back-end, utilizando las API para poder relacionarse y que se puedan comunicar de manera correcta. Debo comentar que fue casi a final del proyecto cuando empecé a mirar como estaba compuesto back-end y cómo se desarrollaban los métodos a través de un software llamada swagger, que sirve para documentar los métodos de las APIs para luego poder producirlo en la aplicación.

Por último, trabajar en este proyecto me ha supuesto, aunque no lo parezca, un gran tiempo de autoaprendizaje al tener que buscar información. En esta situación no existe la ayuda de una figura donde tenga las respuestas o la manera de aconsejarte y decirte que esta bien o no.

16. Competencias técnicas

CTI1.3: Seleccionar, desplegar, integrar y gestionar sistemas de información que satisfagan las necesidades de la organización con los criterios de coste y calidad identificados

Para satisfacer las necesidades del cliente es necesario un control de calidad asignado por el cliente. Además siempre hay que tener en cuenta el coste total de los servicios usados.

CTI3.1: Concebir sistemas, aplicaciones y servicios basados en tecnologías de red, teniendo en cuenta Internet, web, comercio electrónico, multimedia, servicios interactivos y computación ubicua.

Al estar trabajando en una aplicación web, hemos alcanzado esta competencia debido que accedemos a las base de datos de la empresaria bancaria a través de su API. Además, la aplicación tendrá que tener una interacción con el usuario debido que este realizará peticiones con datos que deberá añadir.

CTI3.3: Diseñar, implantar y configurar redes y servicios.

Esta competencia se ha trabajado debido que hemos estado viendo como se podía mejorar los servicios ofrecidos a través de la web. Además, también hemos estado modificando los servicios ofrecidas por la API, intentando mejor la eficacia de las llamadas a estos.

CTI4: Emplear metodologías centradas en el usuario y la organización para el desarrollo, la evaluación y la gestión de aplicaciones y sistemas basados en tecnologías de la información que aseguren la accesibilidad, la ergonomía y la usabilidad de los sistemas

Al ser un proyecto donde tenemos que mejorar tanto la accesibilidad y la usabilidad del sistema, esta competencia la trabajamos en profundidad.

17. Sostenibilidad

17.1. Autoevaluación

Después de realizar el test sobre el conocimiento del tema de la sostenibilidad en la realización de un proyecto, me he dado cuenta que no tenía tanta idea sobre este y que en cualquier proyecto se debería tener en cuenta todos estos aspectos. A medida que iba respondiendo las preguntas, pude entender que se debe visualizar desde la perspectiva de las tres dimensiones: social, económico y medio ambiental.

En segundo lugar, hay muchos conceptos, tales como los efectos ambientales de los productos o técnicas de innovación y generación de ideas, que no tenía en cuenta desde un principio y que debería de tenerlos en cuenta desde un inicio para la mejora de la sostenibilidad, ya que es un tema importante, que nos afecta de un modo u otro, ahora o en un futuro próximo.

Por otro lado, me he dado cuenta que en la universidad, no se ha dado tanta importancia sobre este tema tan relevante y que nos deberían enseñar, no solamente en asignaturas optativas (como es el caso en la actualidad) sino en las asignaturas obligatorias. Es por ello, que durante el transcurso del cuestionario he ido viendo que es un tema principal a la hora de desarrollar el proyecto y que no he tenido una previa formación que me haya ayudado a decidir qué medios,técnicas,... escoger a la hora de gestionar este trabajo. Es cierto que alguna de las ideas que se proponen en el test los he podido solventar pero no con los conocimientos que me ha proporcionado la universidad sino con la lógica propia.

Por último, decir que este formulario me ha abierto de una manera los ojos en todo este tema, no del tecnológico sino de los otros factores que intervienen, que no tenía en cuenta o que no pensaba que iba a tener un gran impacto, sobretodo en el ámbito ambiental.

17.2. Dimensión económica

En los apartados anteriores se ha podido ver el coste, tanto como los costes humanos como los costes de hardware, software y otros tipos, total de la preparación y ejecución de este proyecto. También se ha estimado un presupuesto económico en caso de tener un imprevisto o por las contingencias. Se puede ver que la parte más grande se derriba a los recursos humanos. A parte, hay que comentar que si fuera un miembro con mucha más experiencia, ya que he supuesto que los miembros tengan un nivel parecido al mio, el precio por hora sería mayor

pero a la hora de realizar la tarea sería más óptima y con más experiencia a la hora mejorar la experiencia y la optimización.

La poca experiencia en este campo, se puede ver reflejada en las horas de realizar en el sprint, como he comentado anteriormente, sobre las tareas, ya que otro miembro con más experiencia no debería tener tantas dificultades y terminar lo antes posible. Pero debido a la ideología que tienen en mi empresa, prefieren apostar por un usuario que tenga poca experiencia para formarlo y en la creación de nuevos talentos.

17.3. Dimensión ambiental

Durante todo el periodo del proyecto, se utilizarán portátiles como tal, esto conlleva a que la energía que se utilizará será la eléctrica. Es complicado saber la cantidad de energía que se utilizará, pero intentará que sea la mínima, utilizando los recursos mínimos para este proyecto.

Por otra parte, con este proyecto lo que se intenta es que los usuarios no tengan que desplazarse mediante transporte hasta las oficinas y poder hacer ellos mismos las operaciones necesarias. También se añade la opción de imprimir el documento en pdf, esto conlleva a que se podría guardar en ese mismo formato los documentos y así no tener que malgastar en impresiones.

Finalmente, en lo que respecta al ordenador, una vez un usuario deje su puesto siempre se formateara el portátil haciendo que sea uno nuevo. Así podremos reutilizar el principal recurso que se empleará.

17.4. Dimensión Social

El proyecto se desarrolla con la intención de favorecer al usuario final y al cliente, favoreciendo su experiencia en estas operativas y la mejora de las funcionalidades y optimización.

A nivel personal, me ha ayudado bastante a la hora de ver cómo realizar y desarrollar una página web. También lo que comporta a la utilización de las distintas tecnologías a usar y el tener que realizar las tareas de distintos roles.

Finalmente, poder ver tu trabajo en la página web de la entidad es una satisfacción inmensa, tanto a nivel personal como que todos los usuarios pueden ver las modificaciones o mejoras que he implementado yo mismo. Por otro lado, poder ayudar a mejorar la experiencia de todos los miembros y facilitarles el uso de esta.

18. Referencias

Referencias

[1] Que es un sprint de scrum [en linea]. [Consulta: 19 febrero 2020] Disponible en: https://openwebinars.net/blog/que-es-un-sprint-scrum/

[2] User Stories [en linea]. [Consulta: 19 febrero 2020] Disponible en: https://www.agilealliance.org/glossary/user-stories

[3] Servidores de aplicaciones [en linea]. [Consulta: 19 febrero 2020] Disponible en: http://www.jtech.ua.es/j2ee/2003-2004/abierto-j2ee-2003-2004/sa/sesion1-apuntes.htm

[4] Metodología Scrum [en linea].[Consulta: 19 febrero 2020] Disponible en: https://www.sinnaps.com/blog-gestion-proyectos/metodologia-scrum

[5] 3 razones que demuestran la importancia del buen servicio bancario [en línea] [Consulta: 20 febrero 2020] Disponible en: http://blog.cobiscorp.com/3-razones-que-demuestran-la-importancia-del-buen-servicio-bancario

[6] Las ventajas de la tecnología en las empresas [en línea] [Consulta: 20 febrero 2020] Disponible en: https://economipedia.com/actual/las-ventajas-la-tecnologia-las-empresas.html

[7] ¿Qué es ser un programador web? [en linea] [Consulta: 20 febrero 2020] Disponible en: https://www.tokioschool.com/noticias/que-es-un-programador-web/

[8] Analista Programador [En linea] [Consulta: 20 febrero 2020] Disponible en: https://micarreralaboralenit.wordpress.com/2007/12/05/analistas-programadores-que-hacen-y-que-se-necesita-para-serlo/

[9] Patrones arquitectónicos [en línea] [Consulta: 21 febrero 2020] Disponible en: http://www.lsi.us.es/docencia/get.php?id=5438

[10] Arquitectura en Capa [en línea] [Consulta: 22 febrero 2020] Disponible en: https://www.ecured.cu/Arquitectura_en_Capas

[11] Arquitectura Cliente Servidor [en linea] [Consulta: 22 febrero 2020] Disponible en: https://www.ecured.cu/Arquitectura_Cliente_Servidor

[12] Arquitectura Modelo Vista Controlador [en linea] [Consulta: 22 febrero 2020] Disponible en: https://www.ecured.cu/Patr%C3%B3n_Modelo_Vista_Controlador

[13] ¿Que es Jenkins? Introduccion [en linea] [Consulta: 23 febrero 2020] Disponible en: https://openwebinars.net/blog/que-es-jenkins-introduccion/

Índice de figuras

Listings